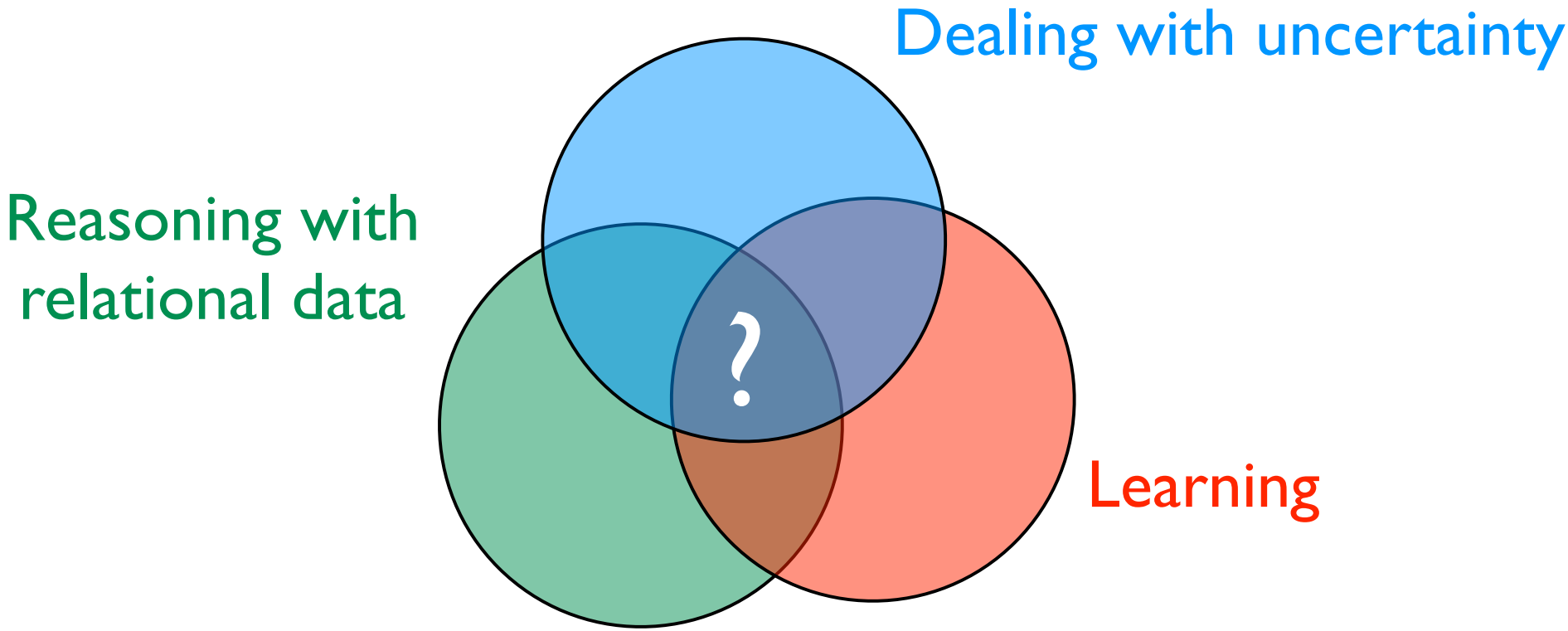
Probabilistic Programming

Luc De Raedt and Angelika Kimmig

ECAI 2014

A key question in AI:

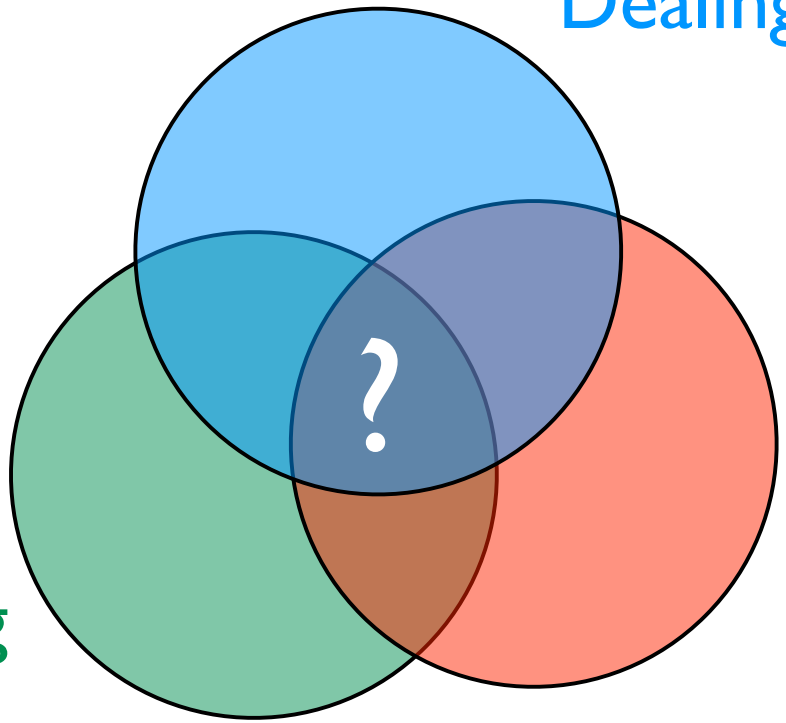


A key question in AI:

Dealing with uncertainty

Reasoning with relational data

- logic
- databases
- programming
- ...



Learning

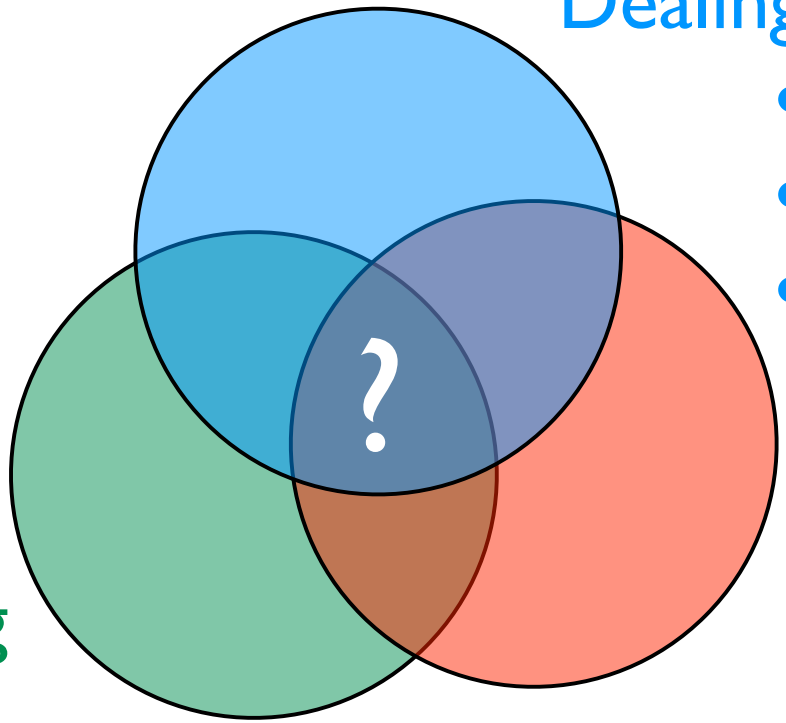
A key question in AI:

Dealing with uncertainty

- probability theory
- graphical models
- ...

Reasoning with relational data

- logic
- databases
- programming
- ...

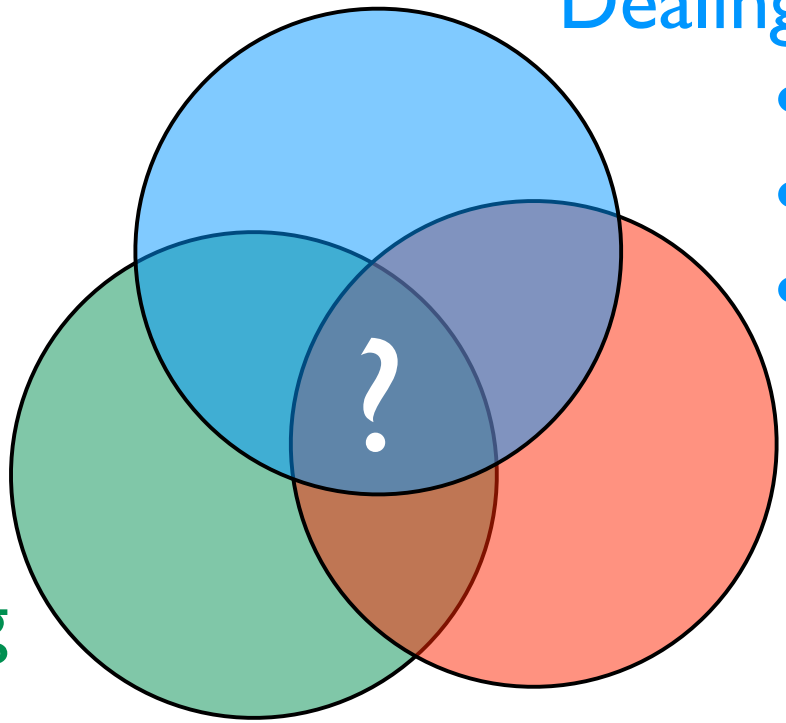


Learning

A key question in AI:

Dealing with uncertainty
- probability theory
- graphical models
- ...

Reasoning with relational data
- logic
- databases
- programming
- ...



Learning
- parameters
- structure

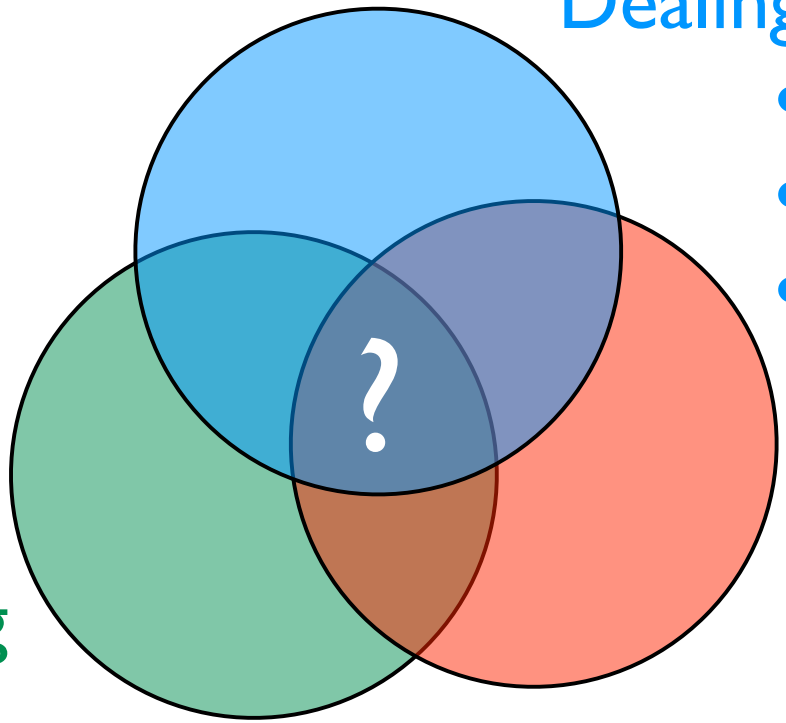
A key question in AI:

Dealing with uncertainty

- probability theory
- graphical models
- ...

Reasoning with relational data

- logic
- databases
- programming
- ...

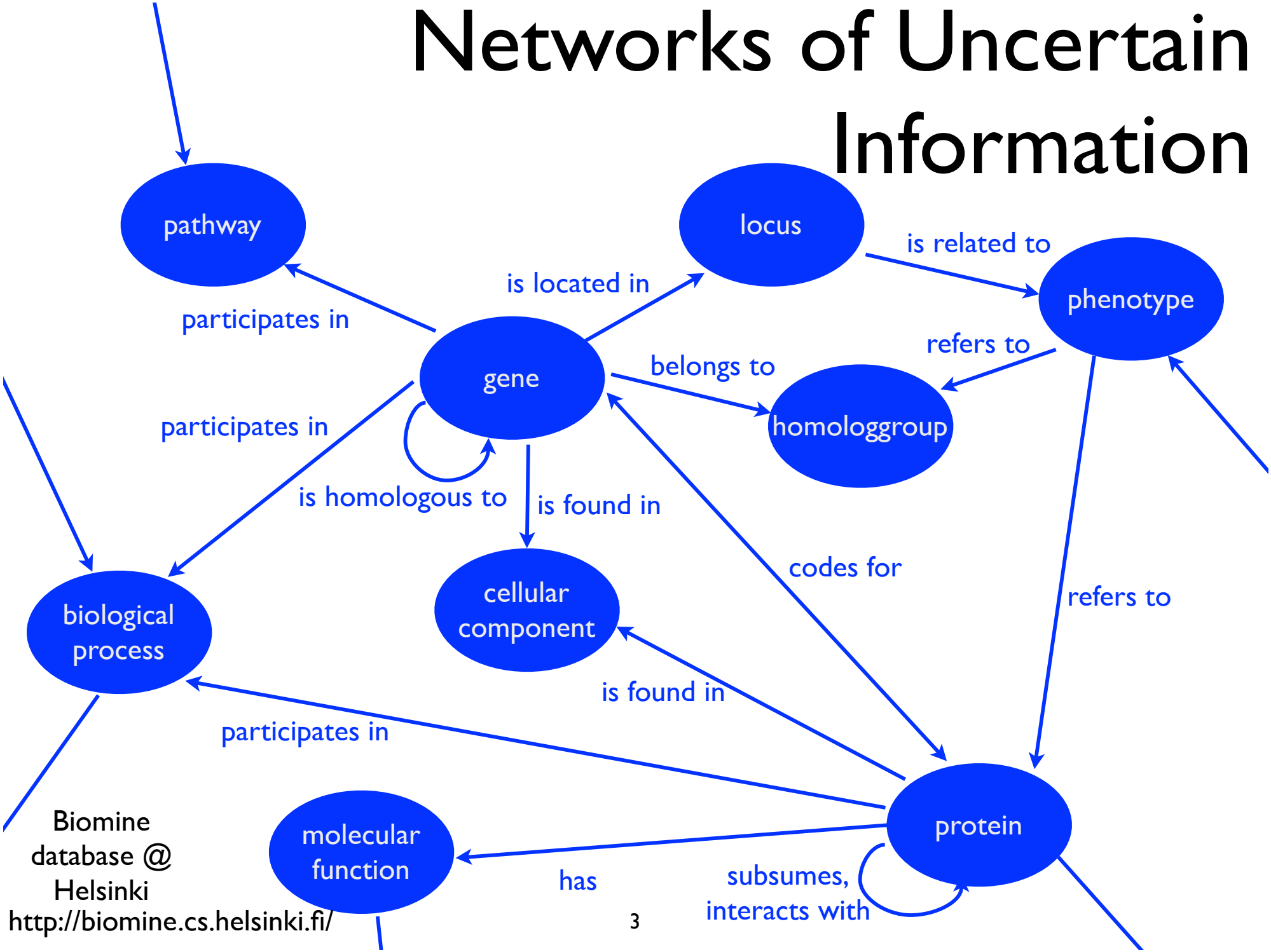


Learning

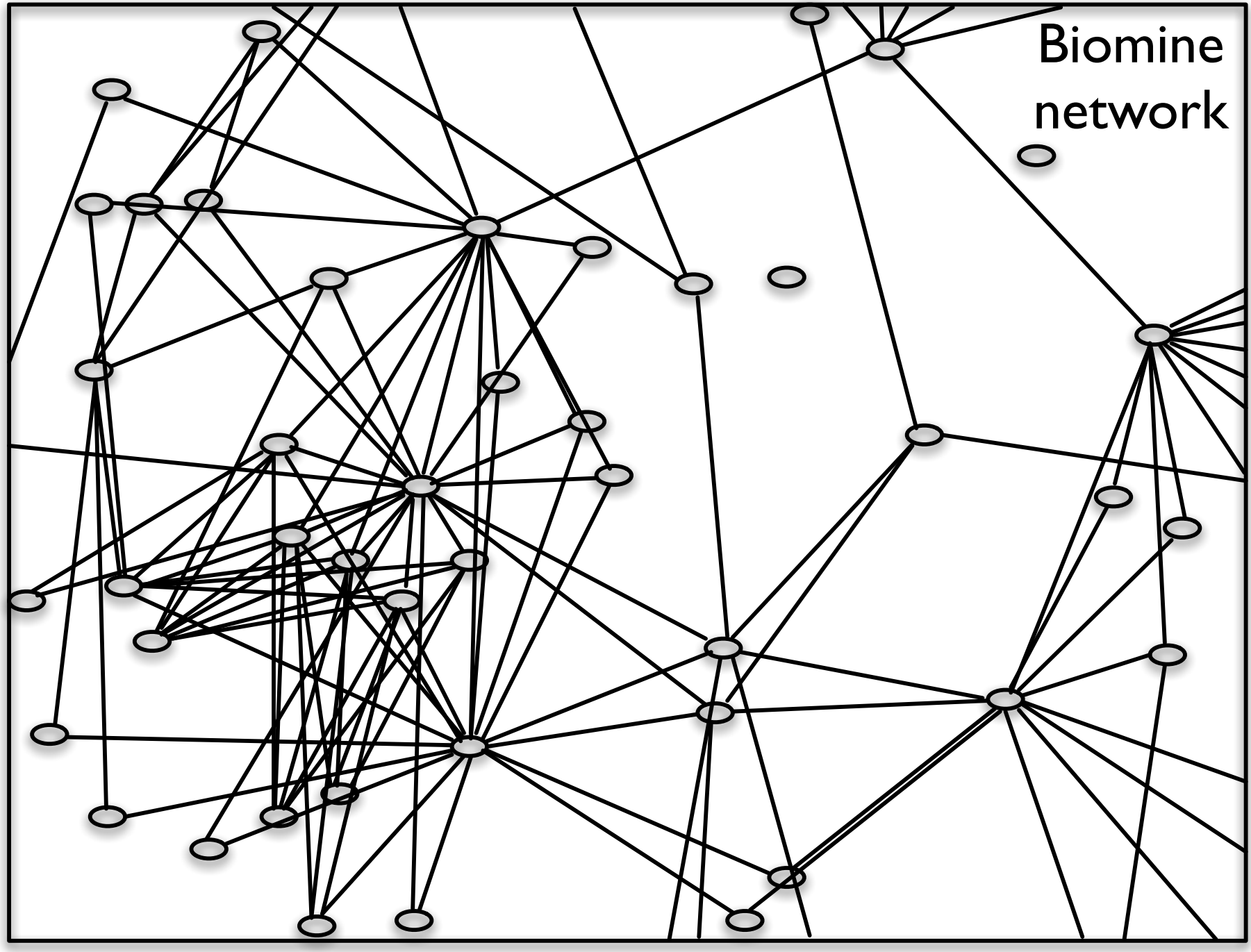
- parameters
- structure

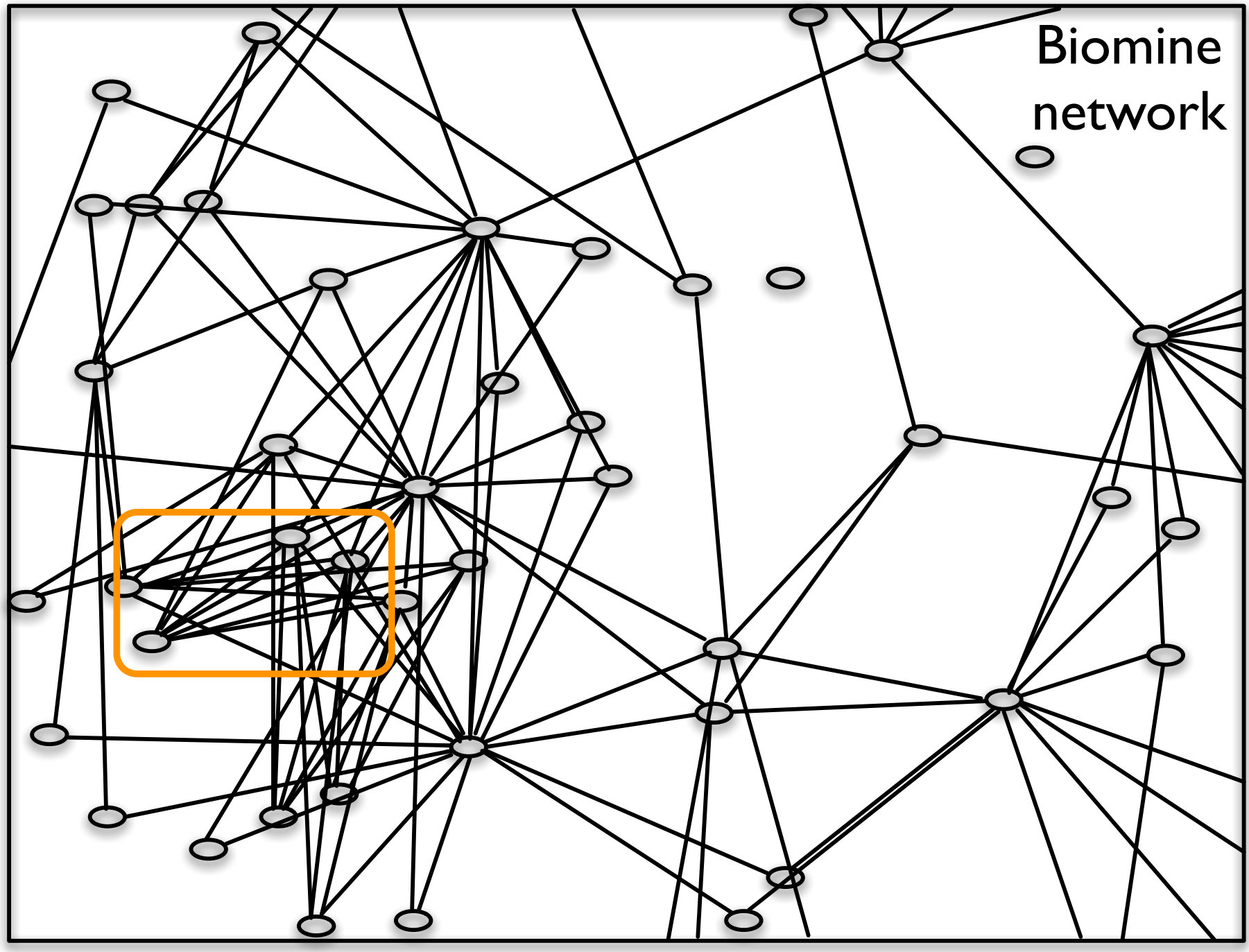
Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

Networks of Uncertain Information

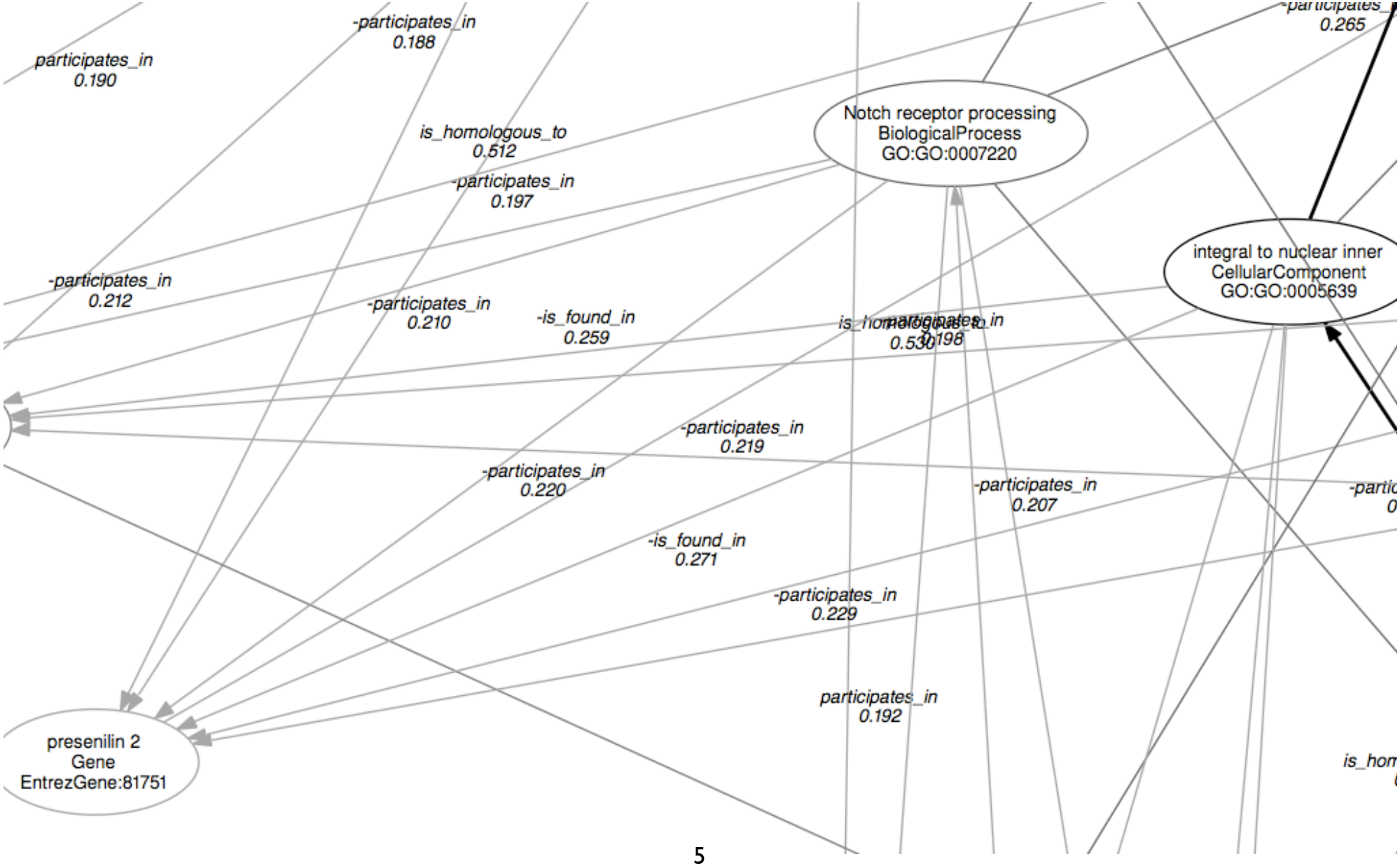


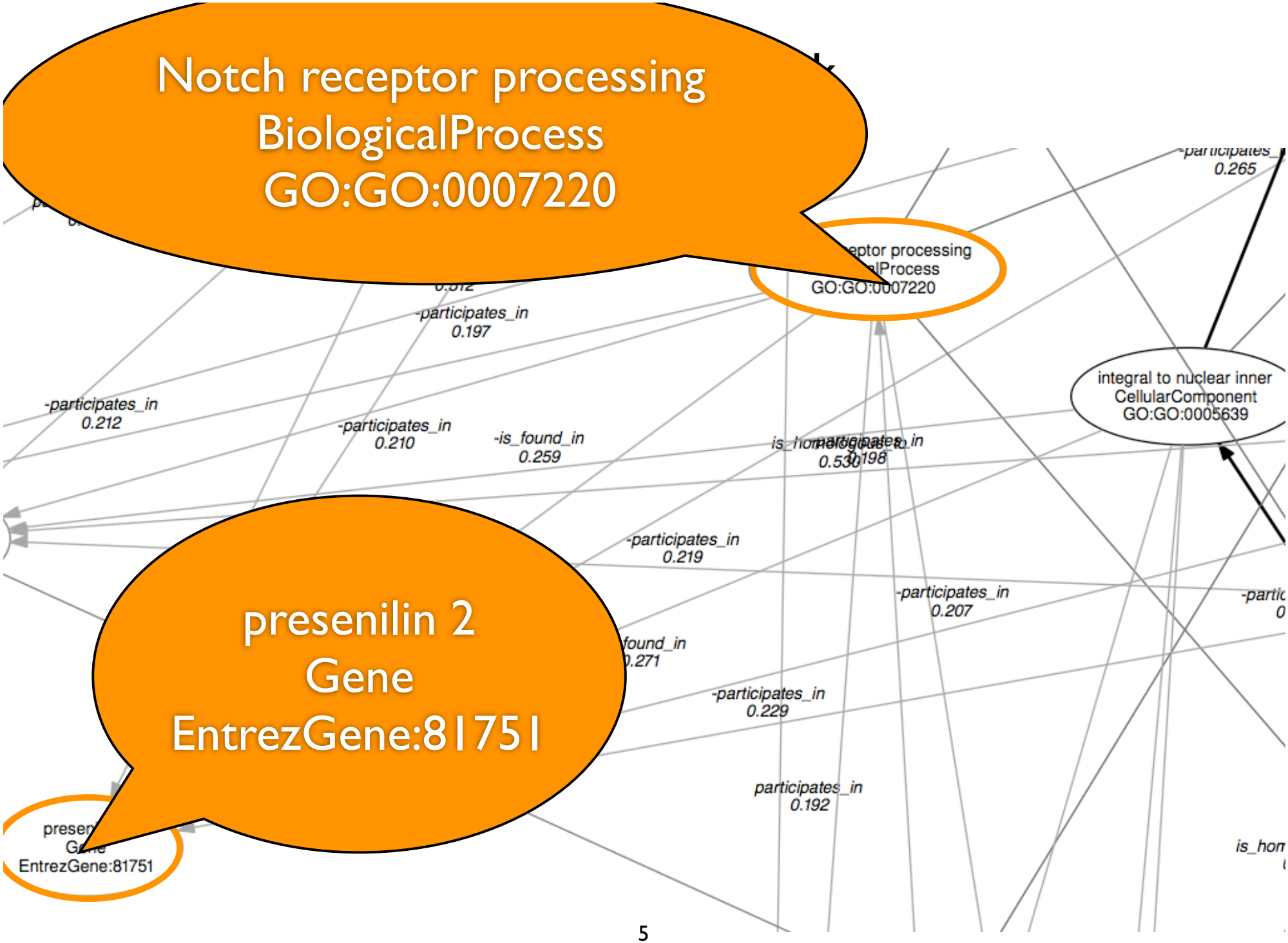
Biomine database @ Helsinki
http://biomine.cs.helsinki.fi/


Biomine
network

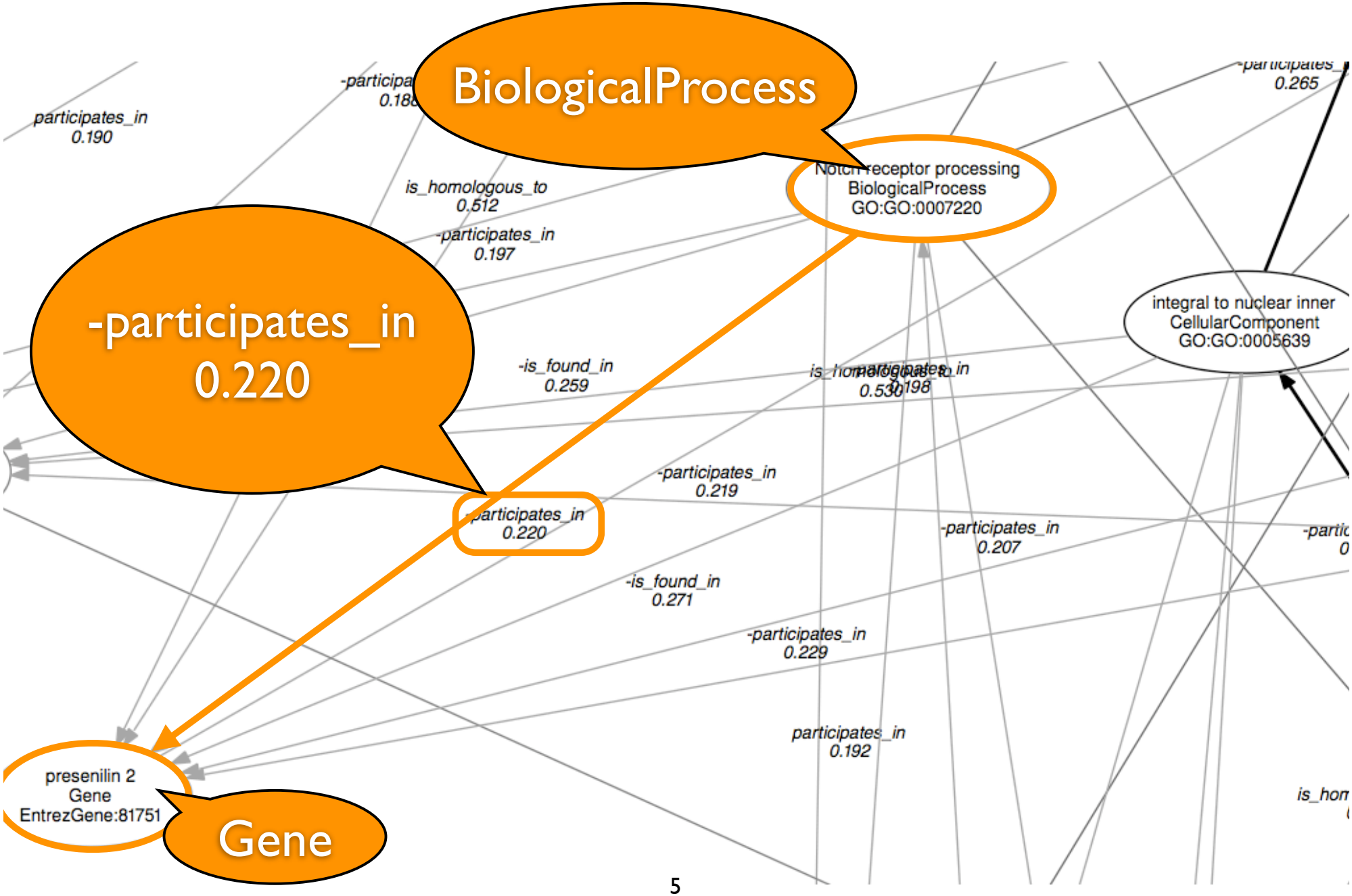

Biomine
network

Biomine Network

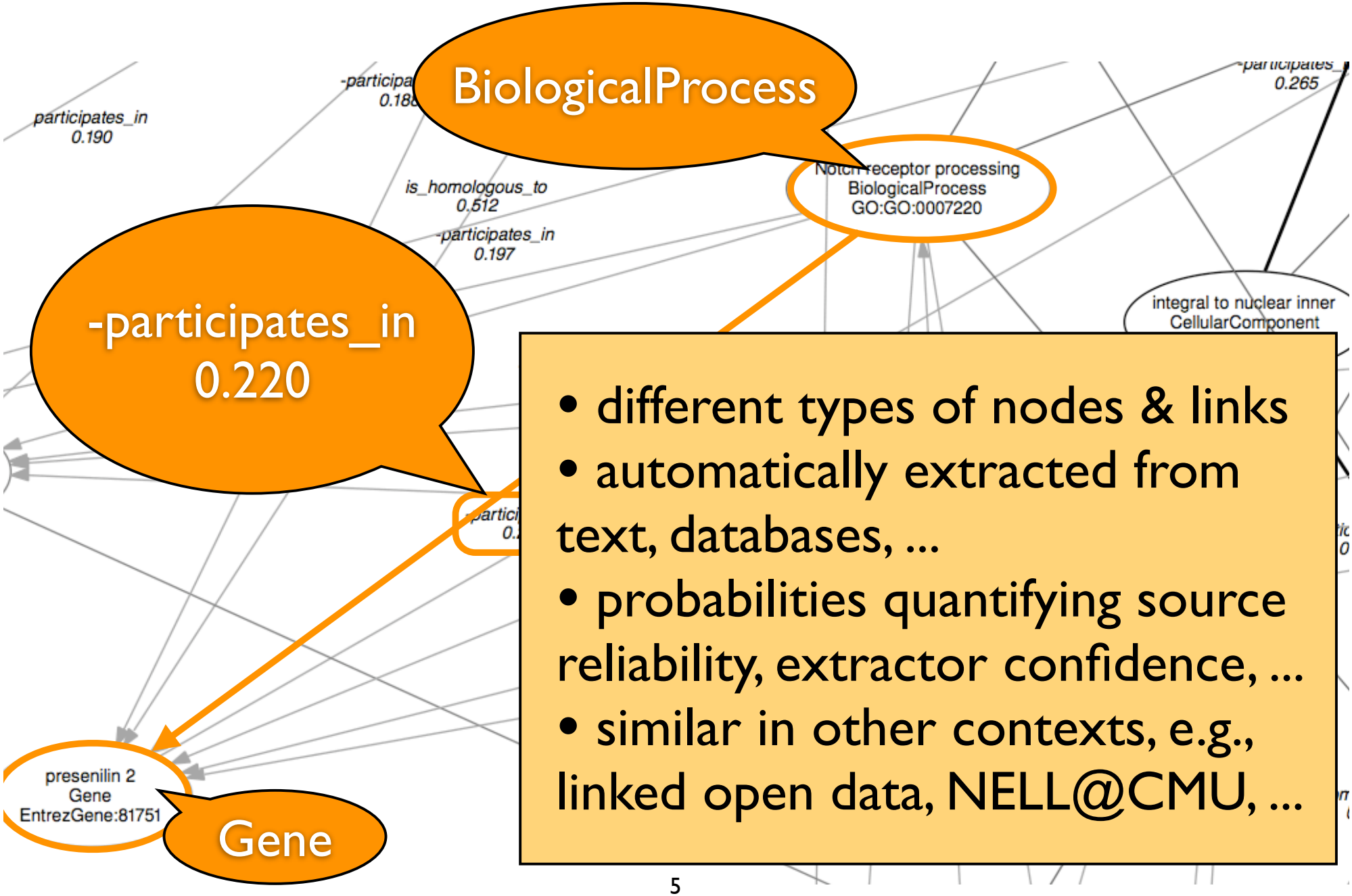



Notch receptor processing
BiologicalProcess
GO:GO:0007220
presenilin 2
Gene
EntrezGene:81751
eptor processing
lProcess
GO:GO:0007220
integral to nuclear inner
CellularComponent
GO:GO:0005639
presen
Gene
EntrezGene:81751
-participates_in
0.265
-participates_in
0.197
-participates_in
0.212
-participates_in
0.210
-is_found_in
0.259
-participates_in
0.219
-participates_in
0.207
found_in
0.271
-participates_in
0.229
participates_in
0.192

Biomine Network

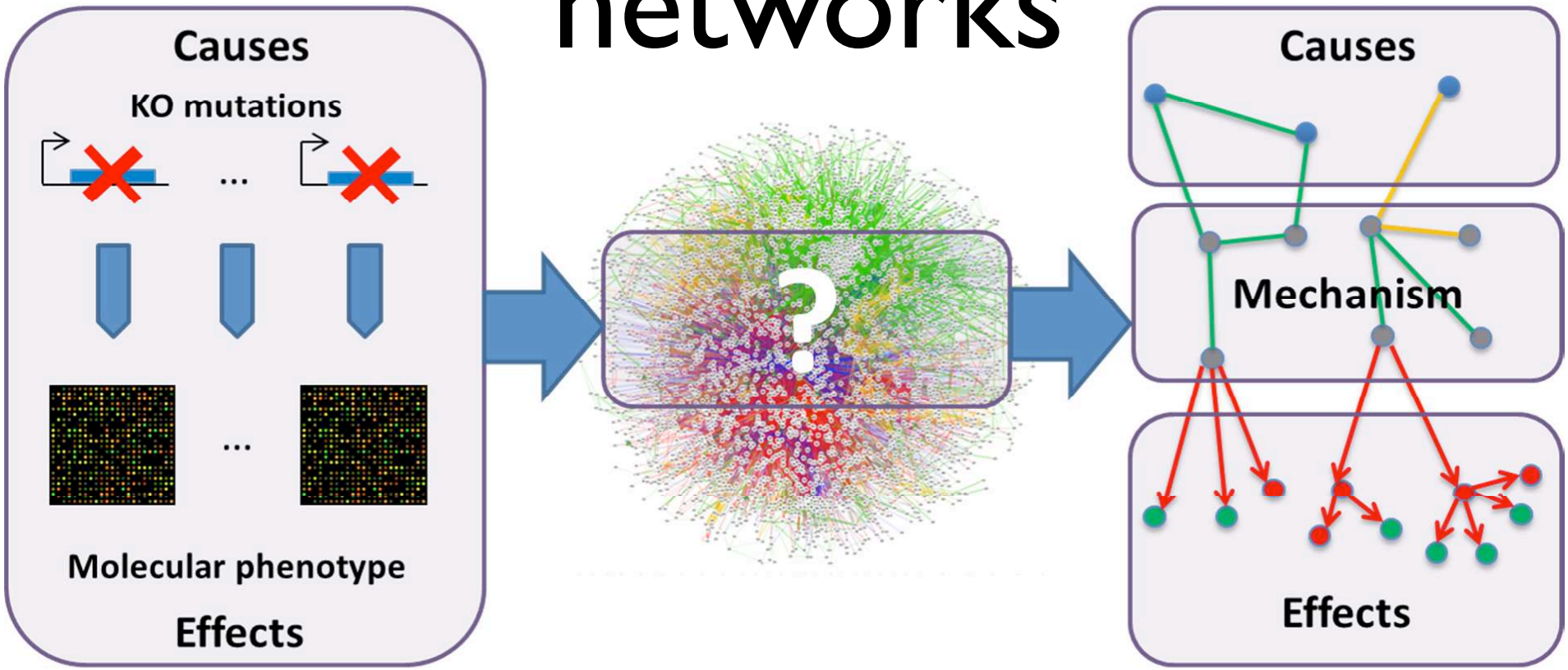


Biomine Network



- different types of nodes & links
- automatically extracted from text, databases, ...
- probabilities quantifying source reliability, extractor confidence, ...
- similar in other contexts, e.g., linked open data, NELL@CMU, ...

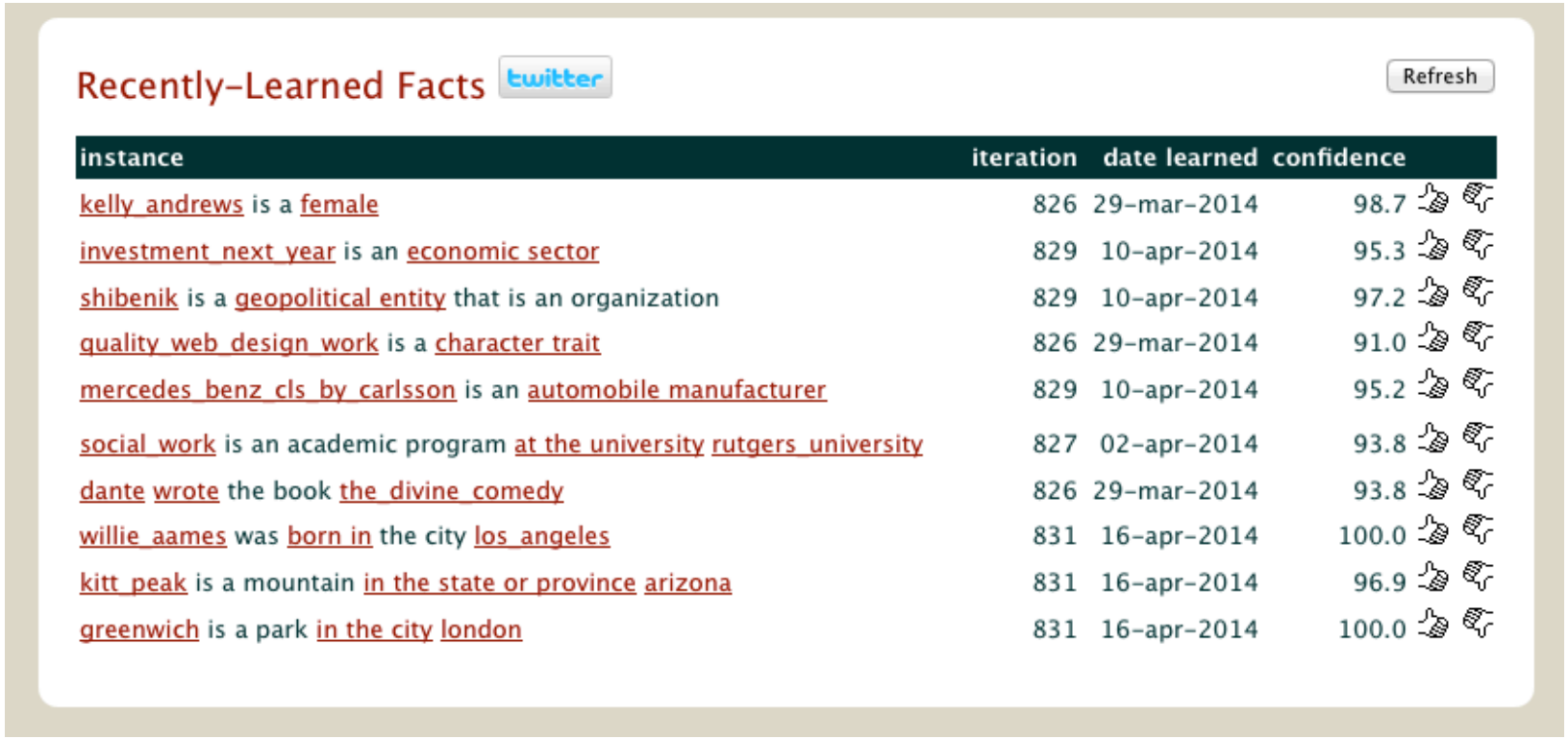
Molecular interaction networks



Can we find the mechanism connecting causes to effects?

[De Maeyer et al, Molecular Biosystems 13]

Example: Information Extraction

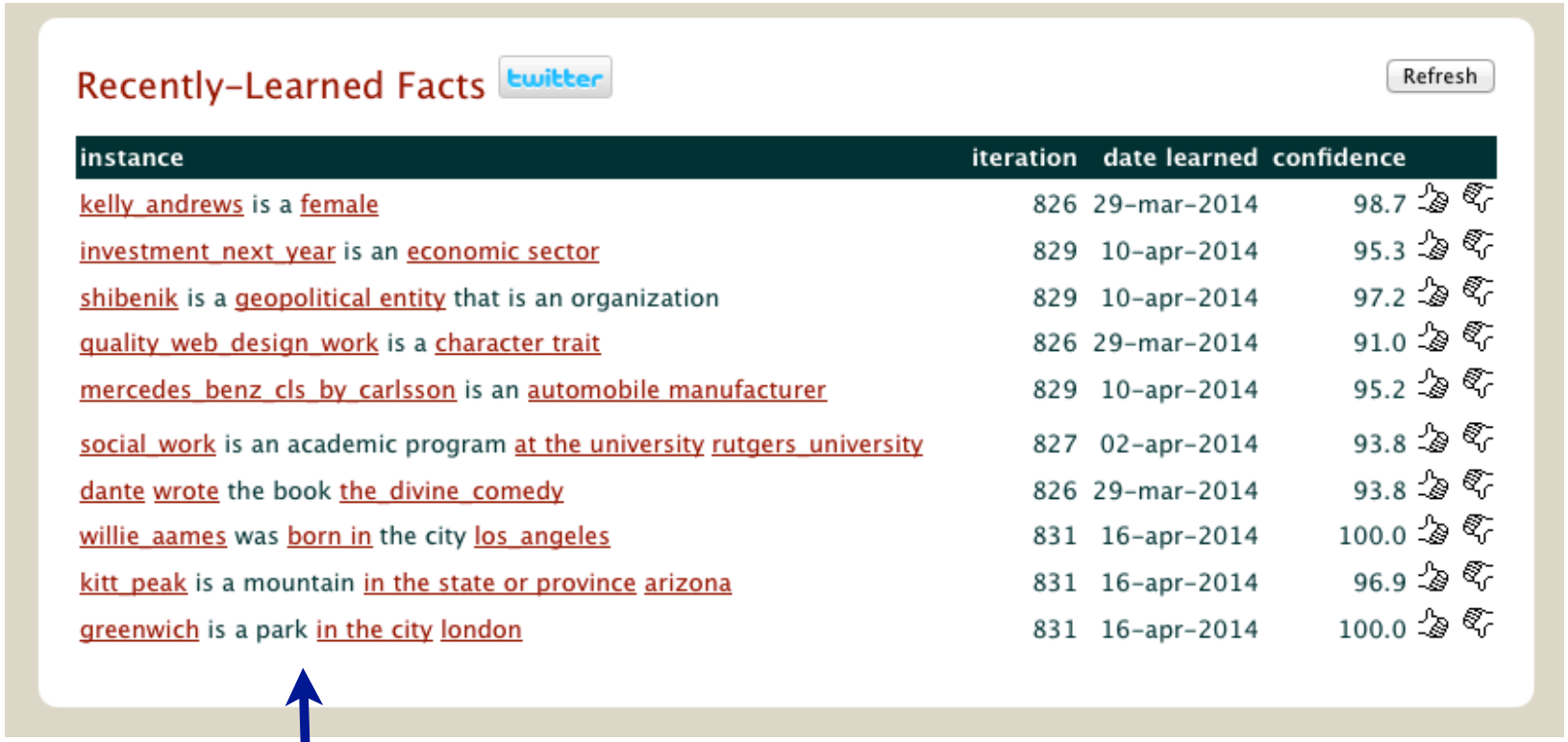


Recently-Learned Facts twitter Refresh

instance	iteration	date learned	confidence
kelly_andrews is a female	826	29-mar-2014	98.7
investment_next_year is an economic sector	829	10-apr-2014	95.3
shibenik is a geopolitical entity that is an organization	829	10-apr-2014	97.2
quality_web_design_work is a character trait	826	29-mar-2014	91.0
mercedes_benz_cls_by_carlsson is an automobile manufacturer	829	10-apr-2014	95.2
social_work is an academic program at the university rutgers_university	827	02-apr-2014	93.8
dante wrote the book the_divine_comedy	826	29-mar-2014	93.8
willie_aames was born in the city los_angeles	831	16-apr-2014	100.0
kitt_peak is a mountain in the state or province arizona	831	16-apr-2014	96.9
greenwich is a park in the city london	831	16-apr-2014	100.0

NELL: http://rtw.ml.cmu.edu/rtw/

Example: Information Extraction



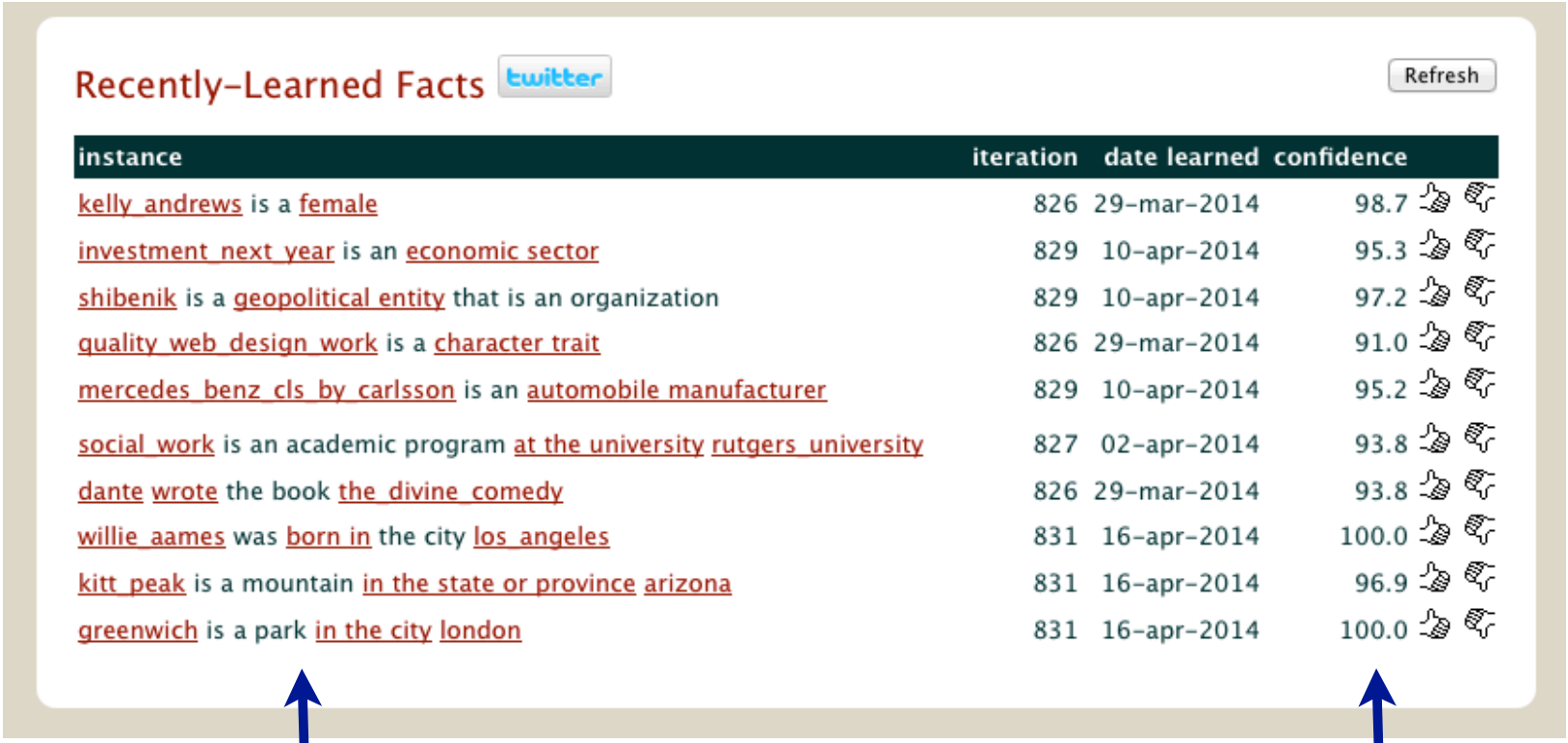
Recently-Learned Facts twitter Refresh

instance	iteration	date learned	confidence
kelly_andrews is a female	826	29-mar-2014	98.7
investment_next_year is an economic sector	829	10-apr-2014	95.3
shibenik is a geopolitical entity that is an organization	829	10-apr-2014	97.2
quality_web_design_work is a character trait	826	29-mar-2014	91.0
mercedes_benz_cls_by_carlsson is an automobile manufacturer	829	10-apr-2014	95.2
social_work is an academic program at the university rutgers_university	827	02-apr-2014	93.8
dante wrote the book the_divine_comedy	826	29-mar-2014	93.8
willie_aames was born in the city los_angeles	831	16-apr-2014	100.0
kitt_peak is a mountain in the state or province arizona	831	16-apr-2014	96.9
greenwich is a park in the city london	831	16-apr-2014	100.0

instances for many different relations

NELL: http://rtw.ml.cmu.edu/rtw/

Example: Information Extraction



Recently-Learned Facts twitter Refresh

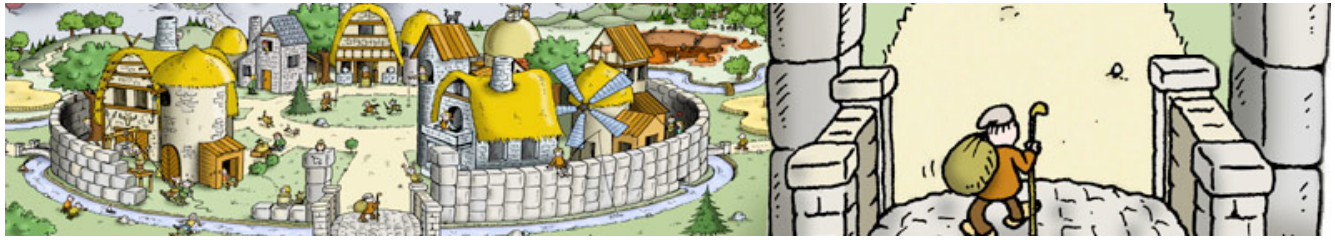
instance	iteration	date learned	confidence
kelly_andrews is a female	826	29-mar-2014	98.7
investment_next_year is an economic sector	829	10-apr-2014	95.3
shibenik is a geopolitical entity that is an organization	829	10-apr-2014	97.2
quality_web_design_work is a character trait	826	29-mar-2014	91.0
mercedes_benz_cls_by_carlsson is an automobile manufacturer	829	10-apr-2014	95.2
social_work is an academic program at the university rutgers_university	827	02-apr-2014	93.8
dante wrote the book the_divine_comedy	826	29-mar-2014	93.8
willie_aames was born in the city los_angeles	831	16-apr-2014	100.0
kitt_peak is a mountain in the state or province arizona	831	16-apr-2014	96.9
greenwich is a park in the city london	831	16-apr-2014	100.0

instances for many different relations

degree of certainty

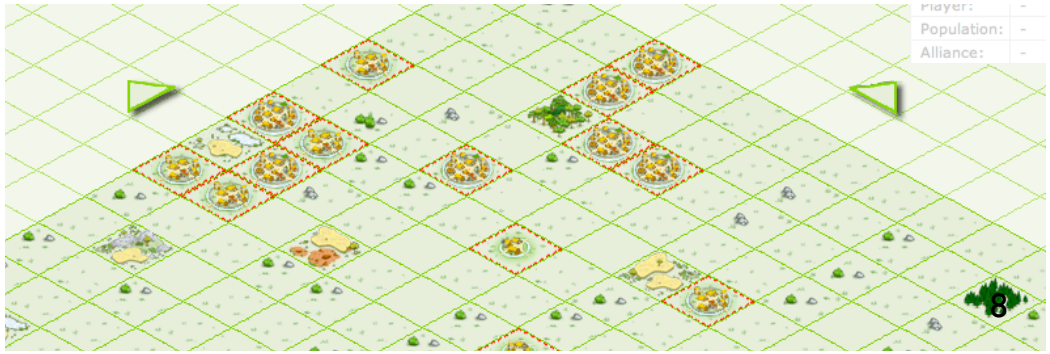
NELL: http://rtw.ml.cmu.edu/rtw/

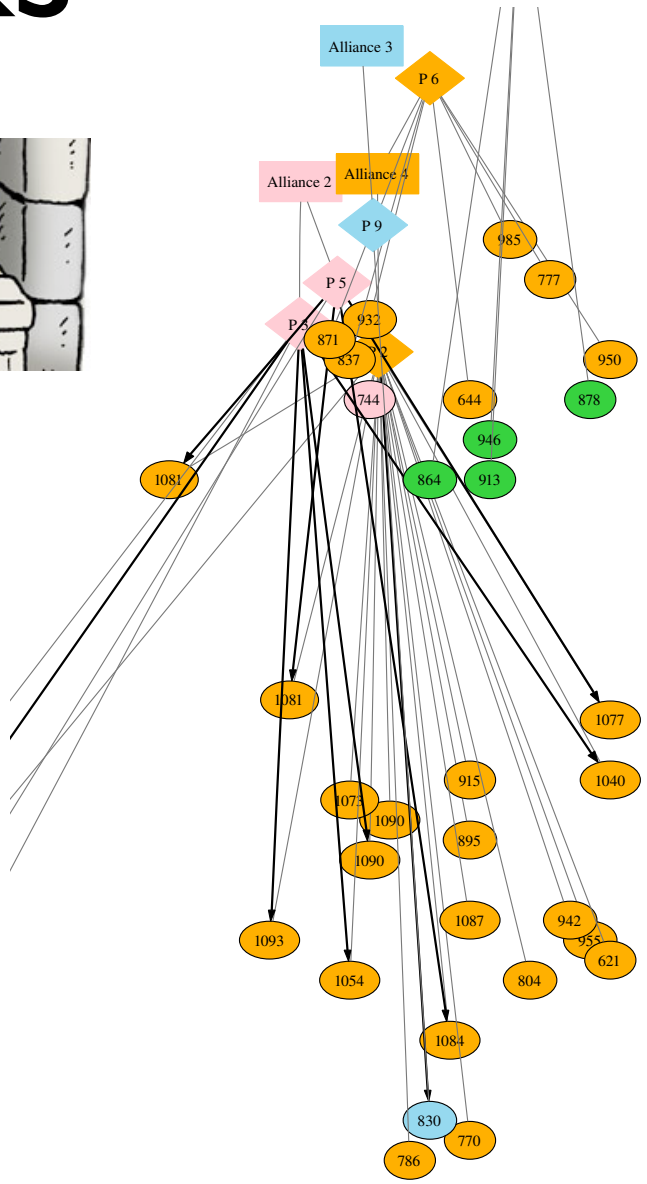
Dynamic networks



Travian: A massively multiplayer real-time strategy game

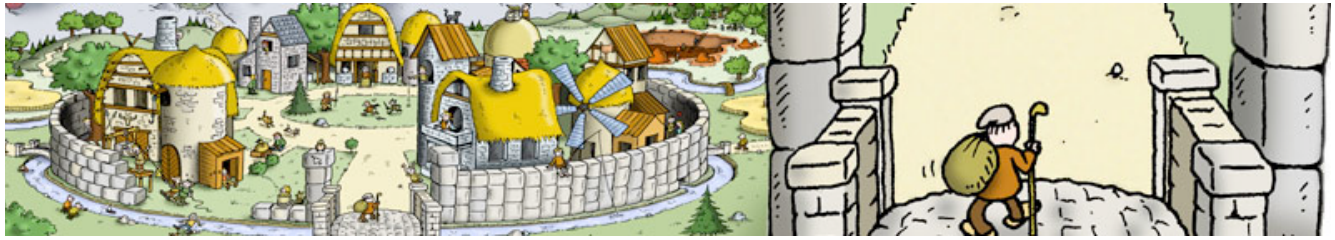
Can we build a model
of this world ?
Can we use it for playing
better ?





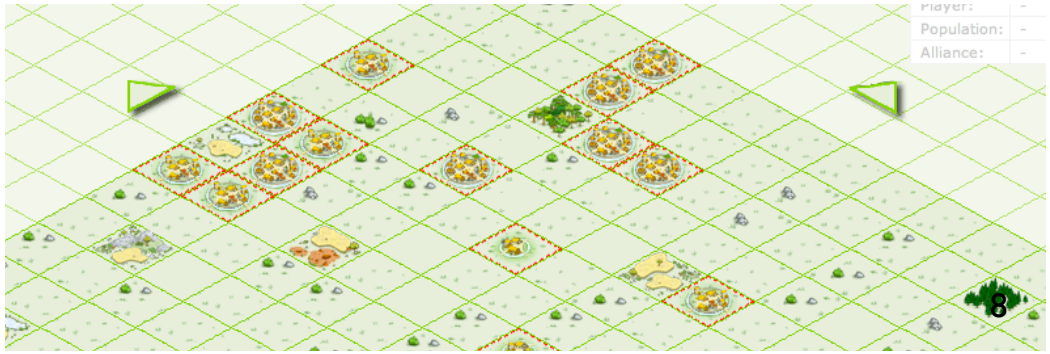
[Thon et al, MLJ 11]

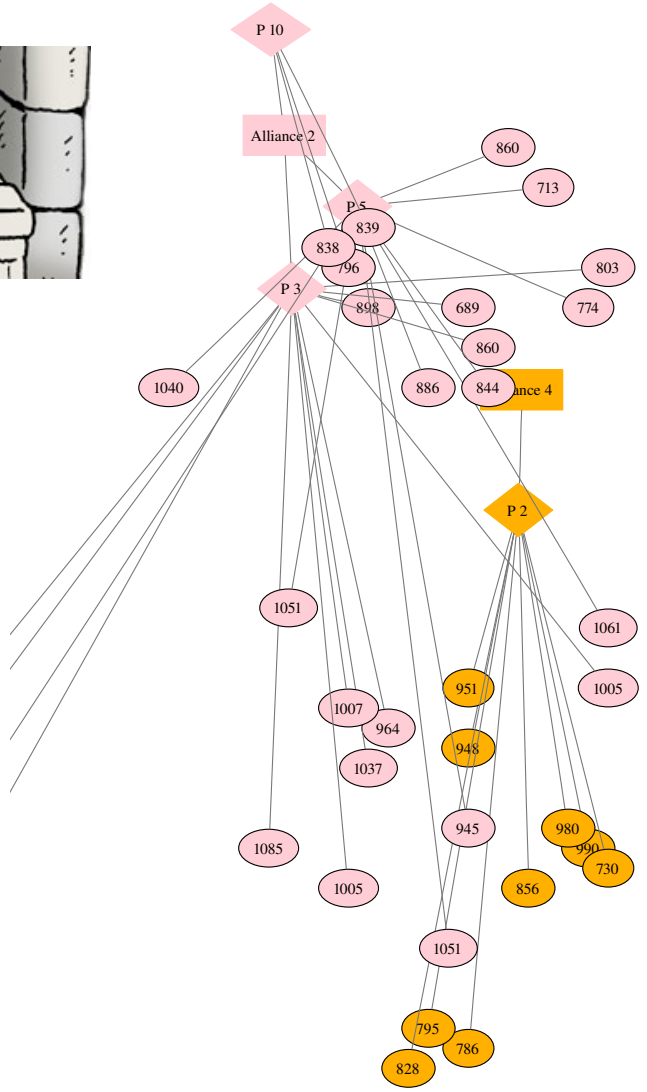
Dynamic networks



Travian: A massively multiplayer real-time strategy game

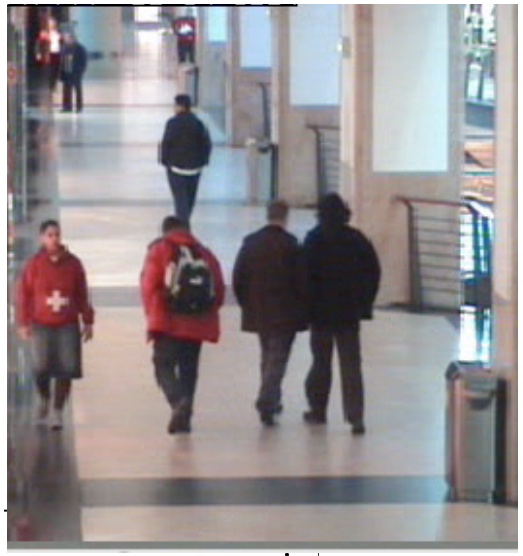
Can we build a model
of this world ?
Can we use it for playing
better ?

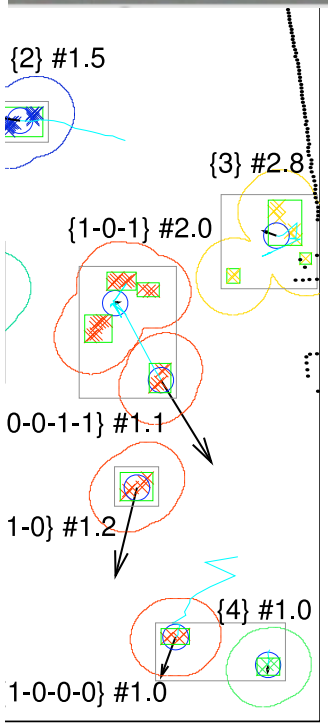




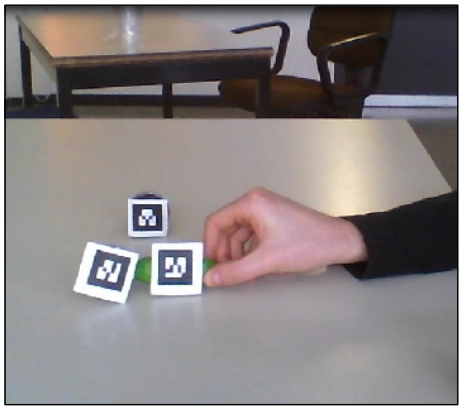
[Thon et al, MLJ 11]

Analyzing Video Data



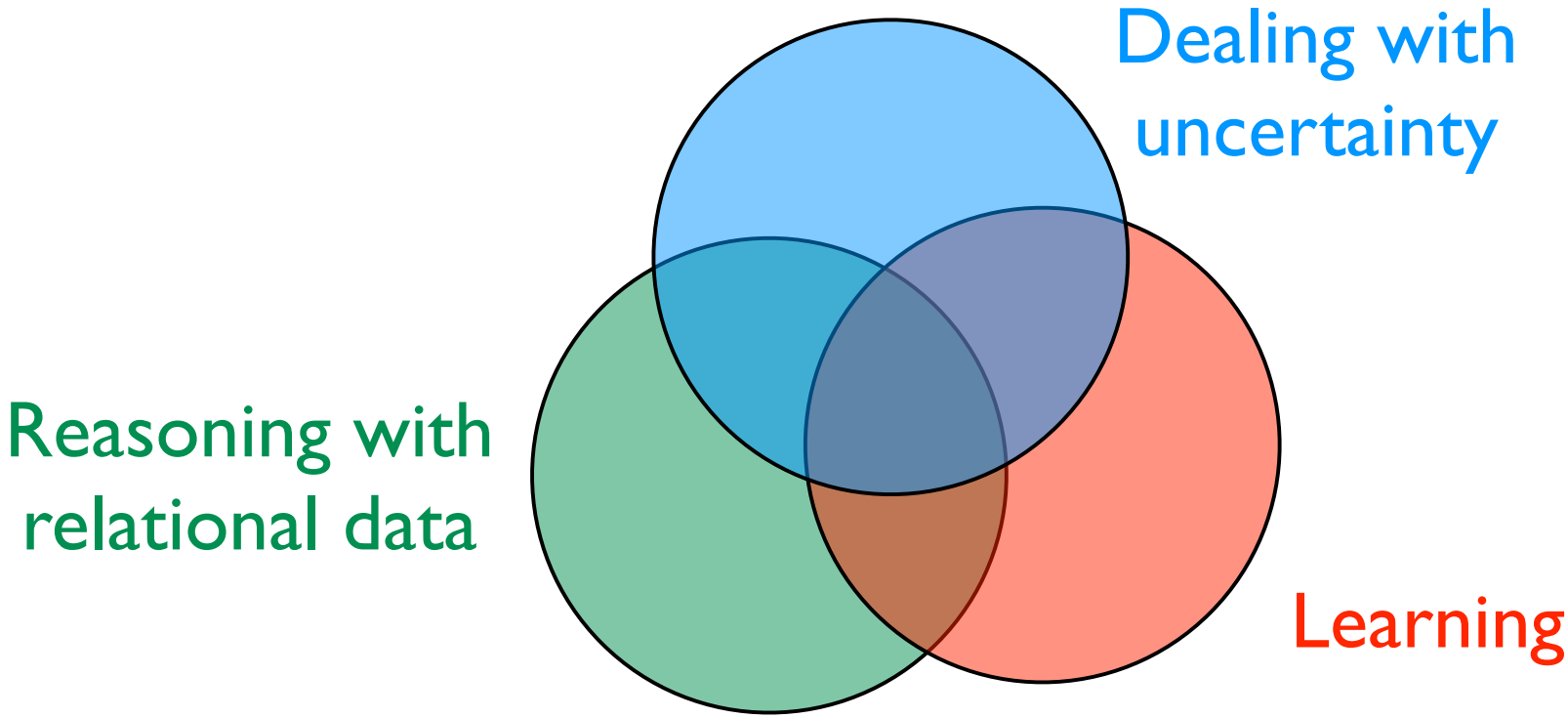


- Track people or objects over time? Even if temporarily hidden?
- Recognize activities?
- Infer object properties?



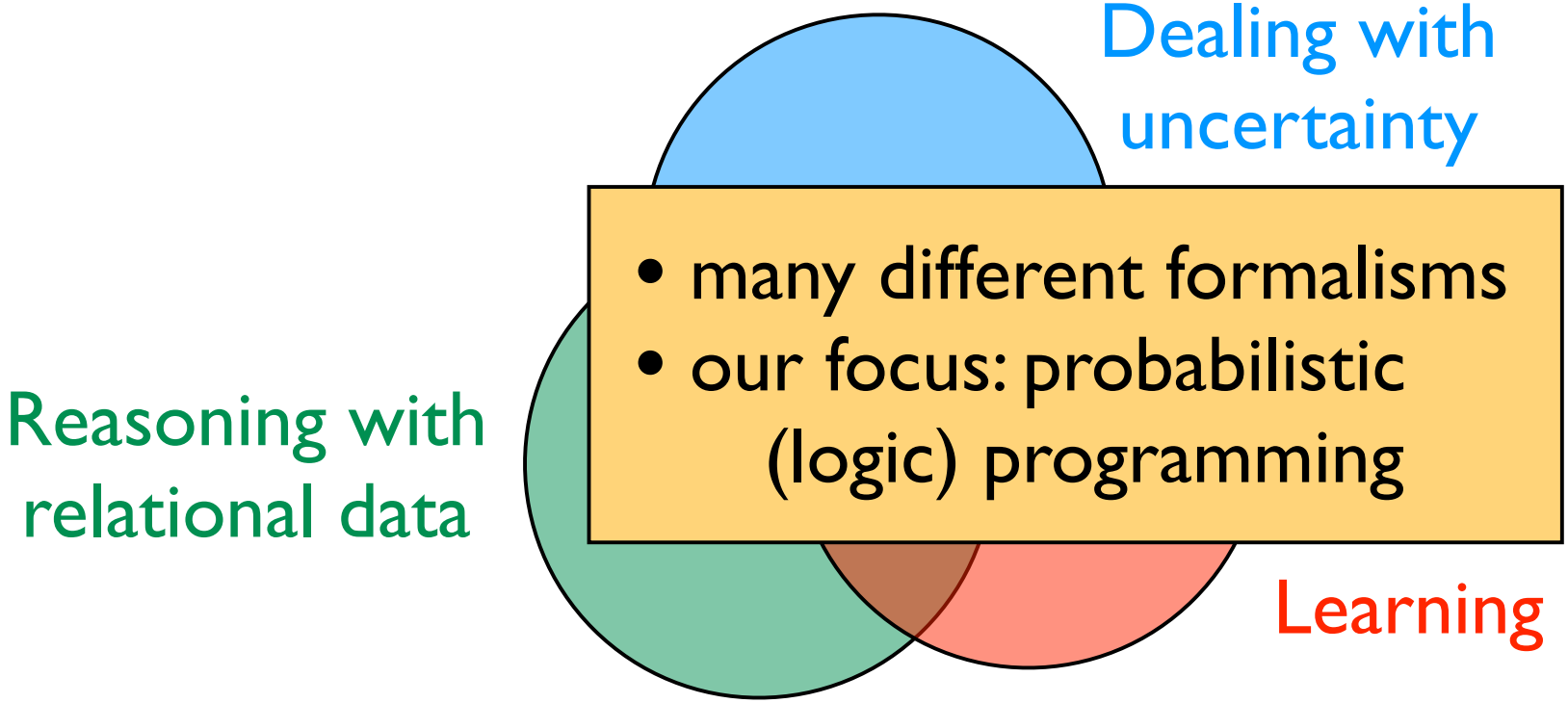
[Skarlatidis et al, TPLP 14;
Nitti et al, IROS 13, ICRA 14]

Common theme



Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

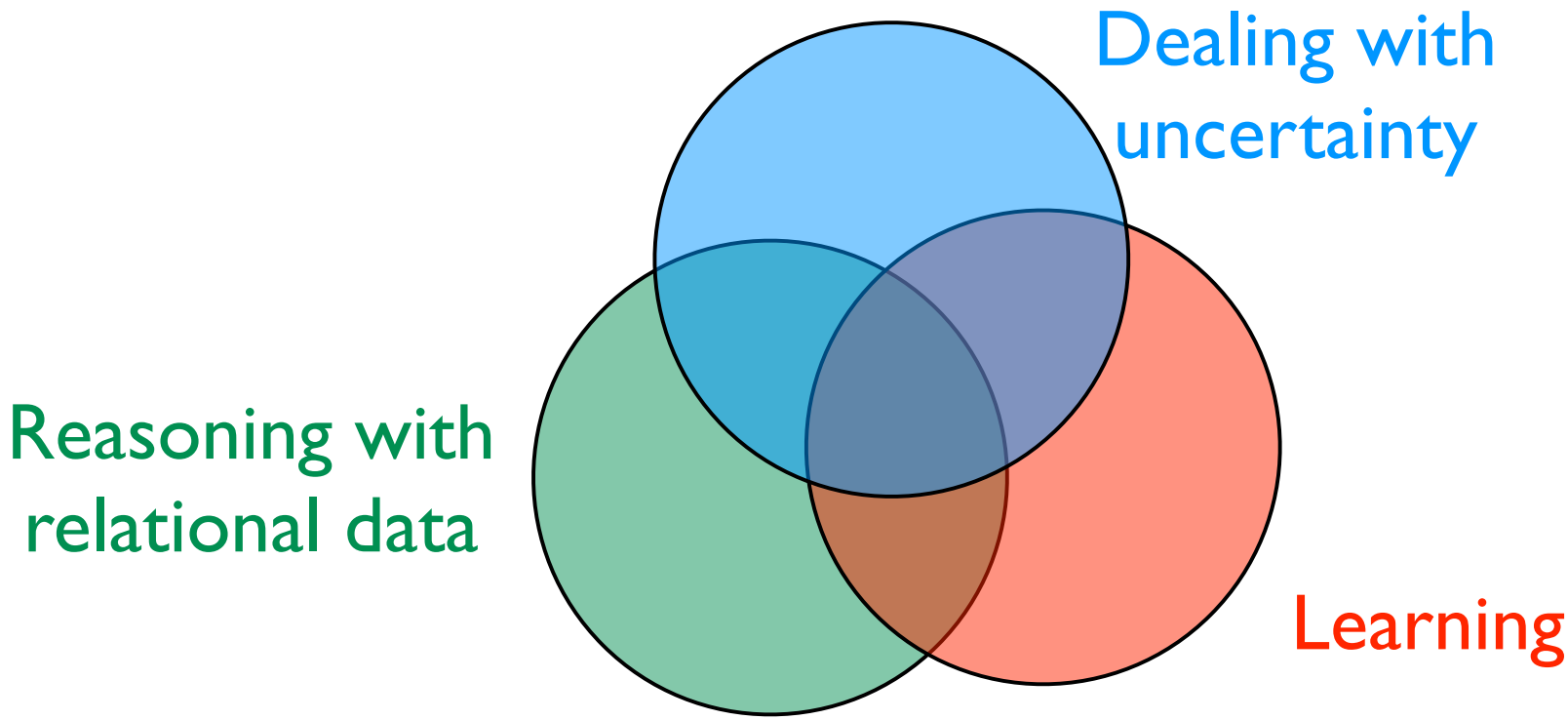
Common theme



Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

ProbLog

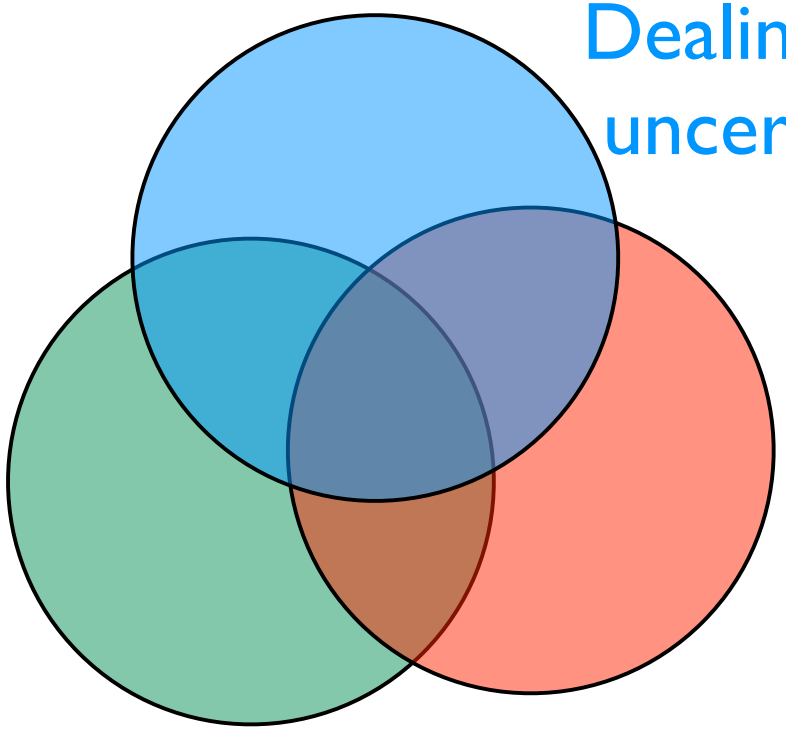
probabilistic Prolog



http://dtai.cs.kuleuven.be/problog/

ProbLog

probabilistic Prolog

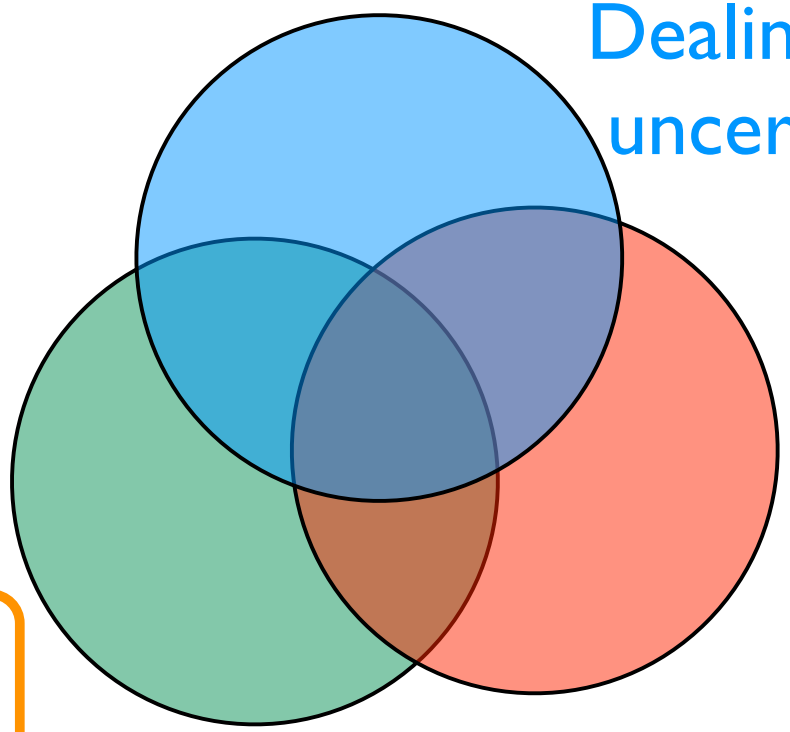



```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```


http://dtai.cs.kuleuven.be/problog/

ProbLog

probabilistic Prolog




```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

one world

```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```

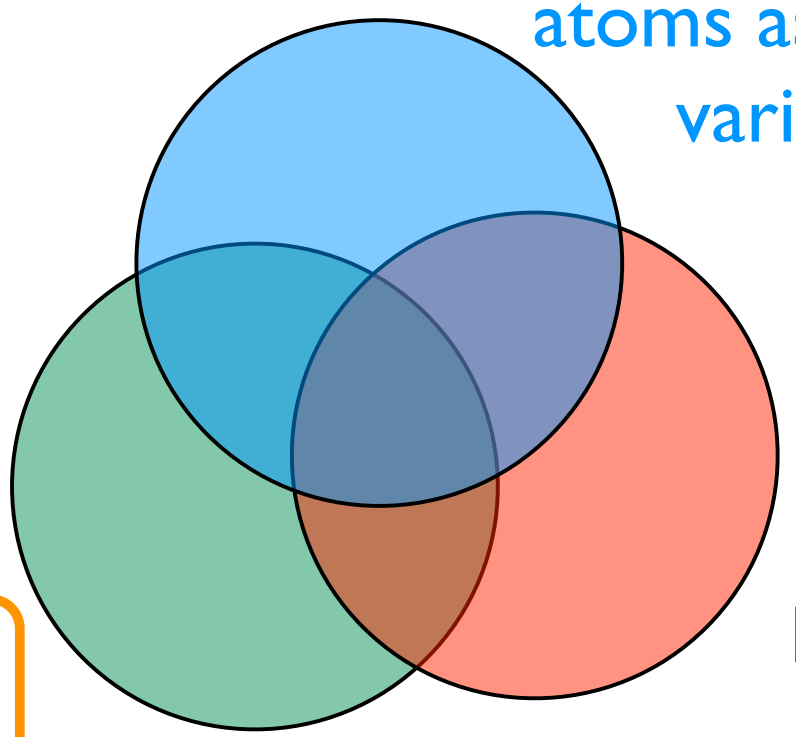

http://dtai.cs.kuleuven.be/problog/

ProbLog

probabilistic Prolog

```
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

atoms as random variables



Prolog / logic programming

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

Learning

one world

```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```


http://dtai.cs.kuleuven.be/problog/

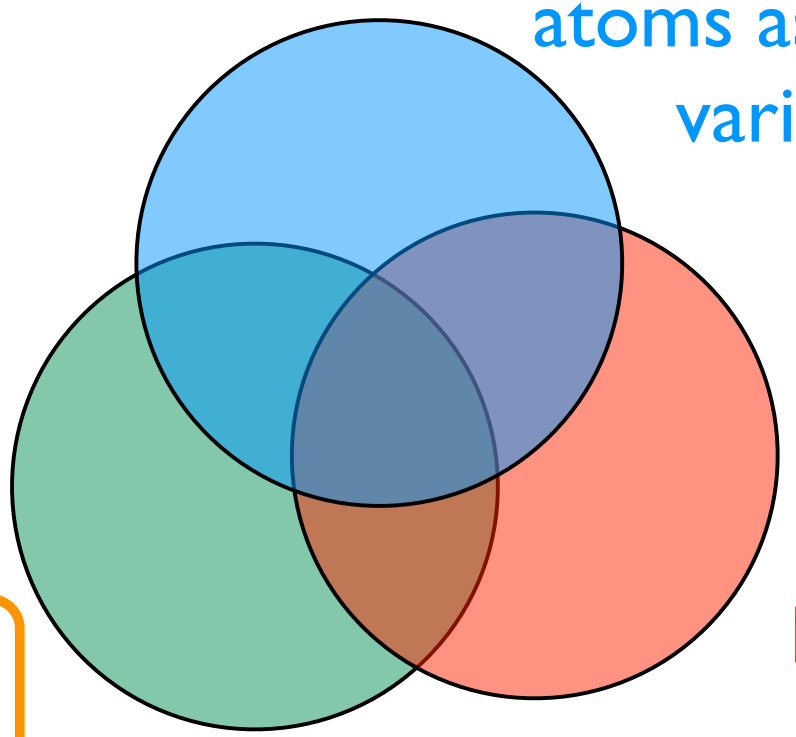
ProbLog

probabilistic Prolog

several possible worlds

```
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

atoms as random variables



Prolog / logic programming

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

Learning

one world

```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```


http://dtai.cs.kuleuven.be/problog/

ProbLog

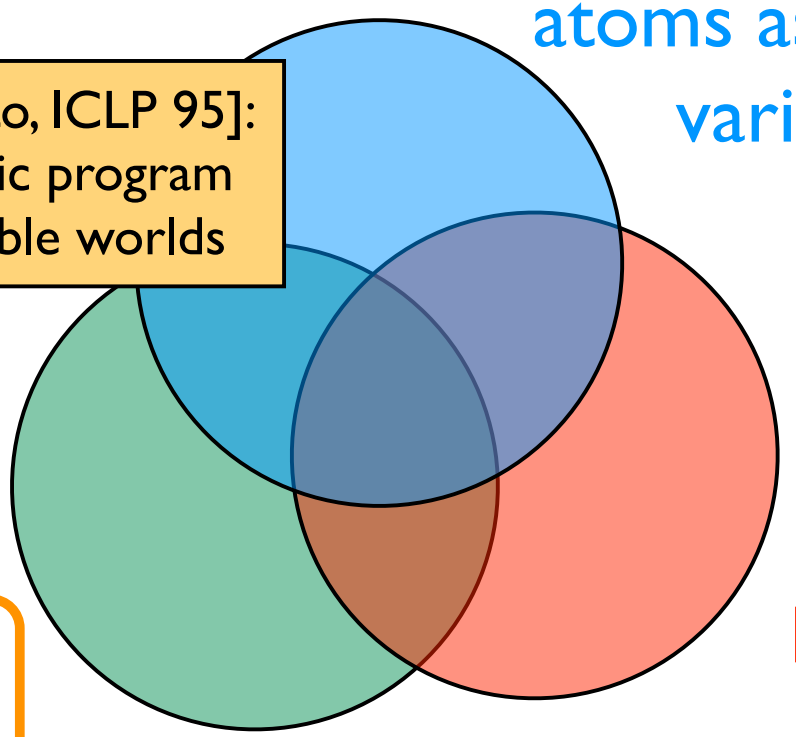
probabilistic Prolog

several possible worlds

```
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

atoms as random variables

Distribution Semantics [Sato, ICLP 95]:
probabilistic choices + logic program
→ distribution over possible worlds



Prolog / logic programming

Learning

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

one world

```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```


http://dtai.cs.kuleuven.be/problog/

ProbLog

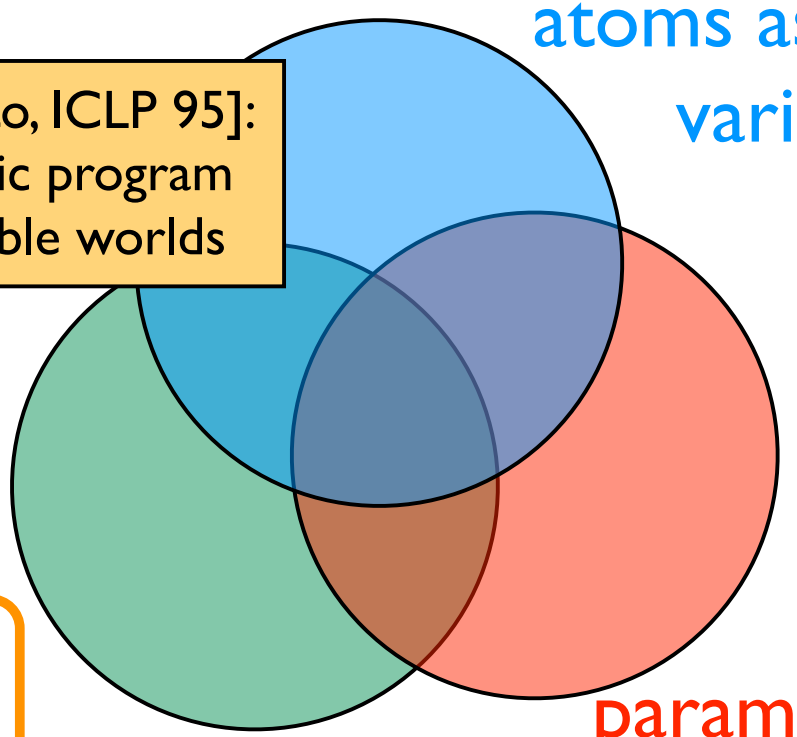
probabilistic Prolog

several possible worlds

```
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

atoms as random variables

Distribution Semantics [Sato, ICLP 95]:
probabilistic choices + logic program
→ distribution over possible worlds



Prolog / logic programming

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

one world

```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```

parameter learning, adapted relational learning techniques

http://dtai.cs.kuleuven.be/problog/

Probabilistic Prologs: Two Views

- Distribution semantics:
  - probability distribution over interpretations
  - degree of belief
- Stochastic Logic Programs (SLPs):
  - probability distribution over query answers
  - like in probabilistic grammars

Graphs & Randomness

ProbLog, Phenetic, Prism, ICL, Probabilistic Databases, ...

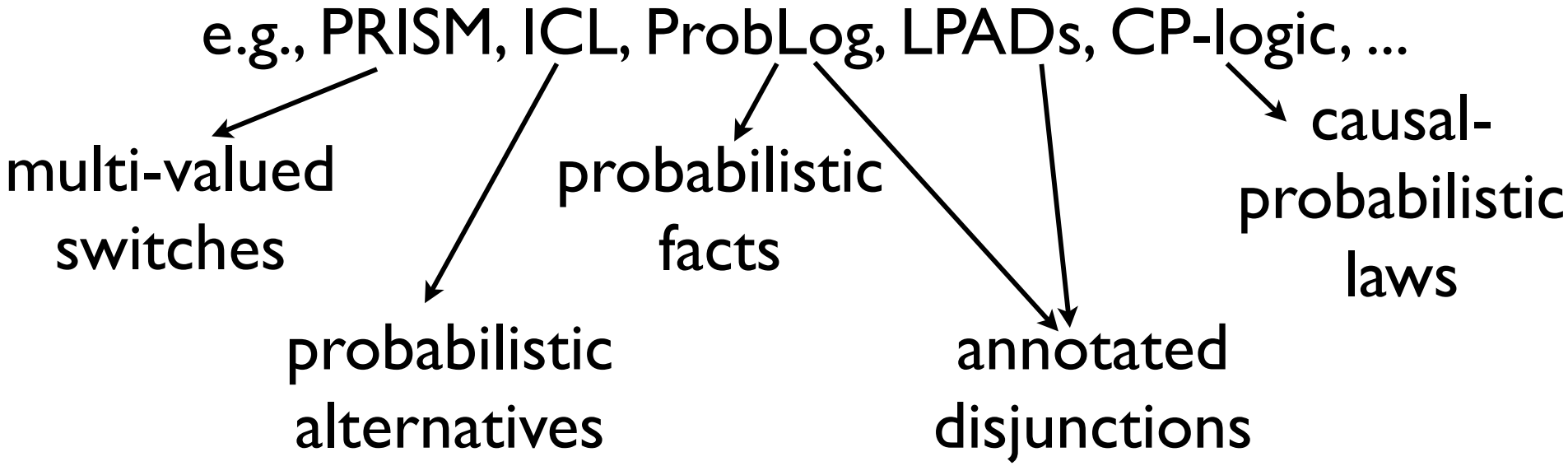
- all based on a "random graph" model

Stochastic Logic Programs, ProPPR, PCFGs, ...

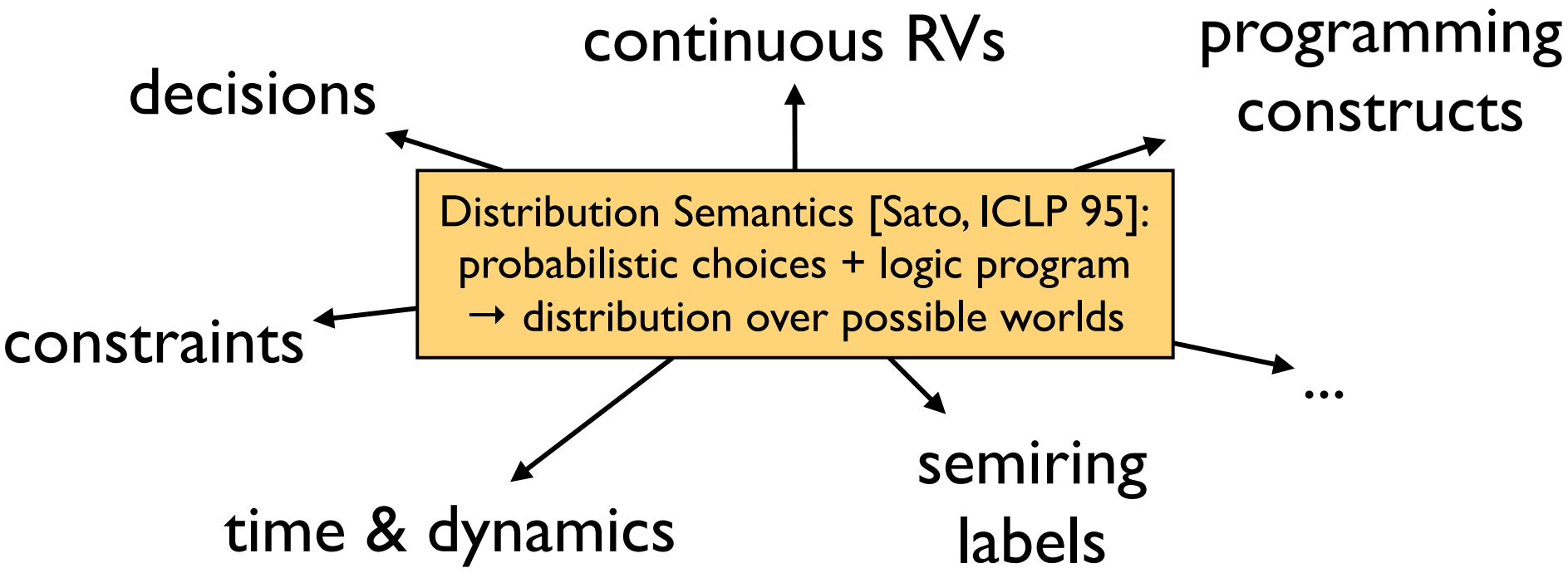
- based on a "random walk" model
- connected to PageRank

Probabilistic Logic Programming

Distribution Semantics [Sato, ICLP 95]:
probabilistic choices + logic program
→ distribution over possible worlds



Extensions of basic PLP



Roadmap

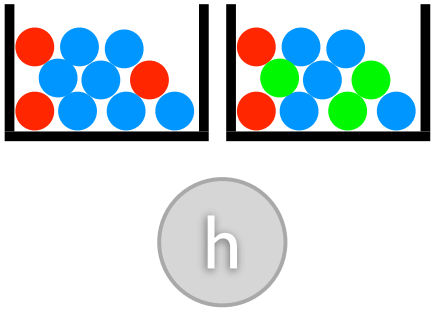
- Modeling
- Reasoning
- Language extensions
- Advanced topics

... with some detours on the way

Part 1 : Modeling

ProbLog by example:

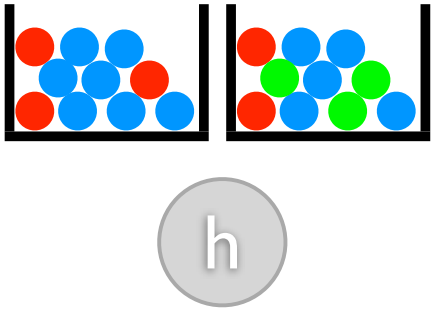
A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

ProbLog by example:

A bit of gambling



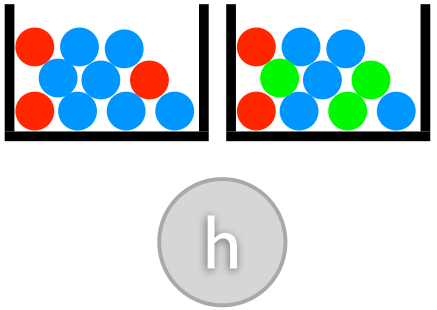
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.
```

probabilistic fact: heads is true with probability 0.4 (and false with 0.6)

ProbLog by example:

A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

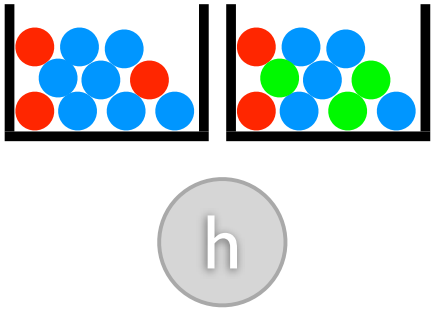
```
0.4 :: heads.
```

annotated disjunction: first ball is red with probability 0.3 and blue with 0.7

```
0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
```

ProbLog by example:

A bit of gambling



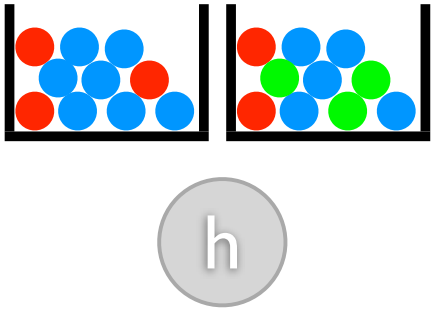
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green);
                   0.5 :: col(2,blue) <- true.
```

annotated disjunction: second ball is red with probability 0.2, green with 0.3, and blue with 0.5

ProbLog by example:

A bit of gambling



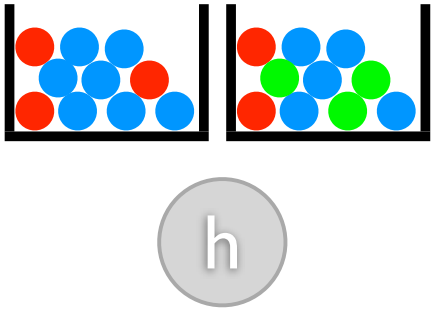
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green);
                   0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
```

logical rule encoding background knowledge

ProbLog by example:

A bit of gambling



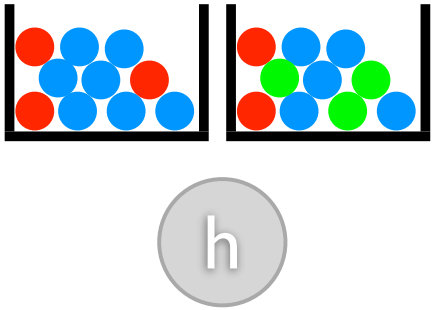
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green);
                   0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

logical rule encoding background knowledge

ProbLog by example:

A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green);
                   0.5 :: col(2,blue) <- true.
```

probabilistic choices

```
win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

consequences

Questions

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

- Probability of **`win`**?
- Probability of **`win`** given **`col(2,green)`**?
- Most probable world where **`win`** is true?

Questions

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

marginal probability

- Probability of **`win`**?

query

- Probability of **`win`** given **`col(2,green)`**?
- Most probable world where **`win`** is true?

Questions

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

marginal probability

- Probability of **`win`**?

conditional probability

- Probability of **`win`** given **`col(2,green)`**?

evidence

- Most probable world where **`win`** is true?

Questions

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

marginal probability

- Probability of **`win`**?

conditional probability

- Probability of **`win`** given **`col(2,green)`**?

- Most probable world where **`win`** is true?

MPE inference

Possible Worlds

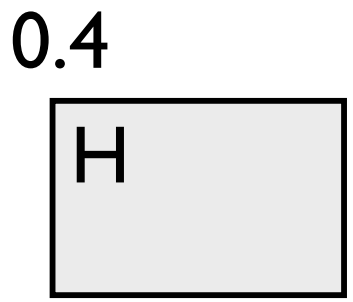
```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

Possible Worlds

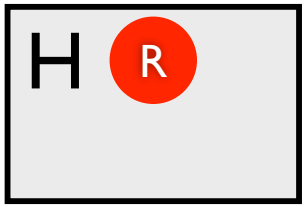
```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```



Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

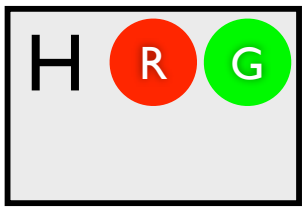
0.4 ×0.3



Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

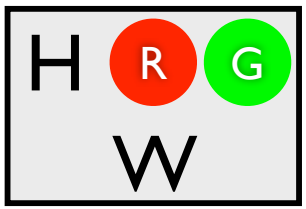
0.4 ×0.3 ×0.3



Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

0.4 ×0.3 ×0.3

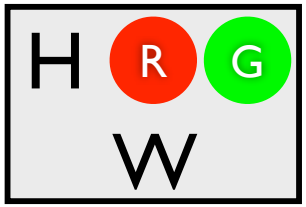


Possible Worlds

```
0.4 :: heads.
```

```
0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$

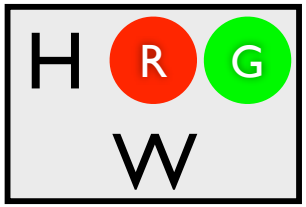


$(1-0.4)$

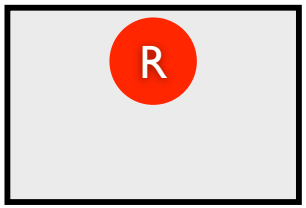
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$



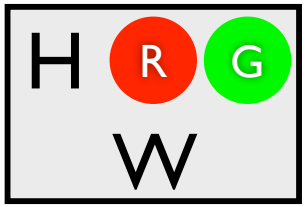
$(1-0.4) \times 0.3$



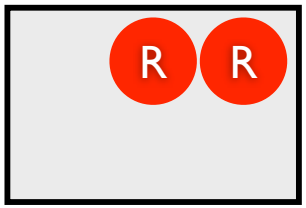
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$



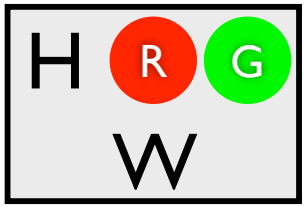
$(1-0.4) \times 0.3 \times 0.2$



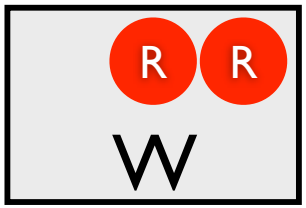
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$

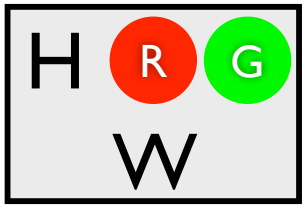


Possible Worlds

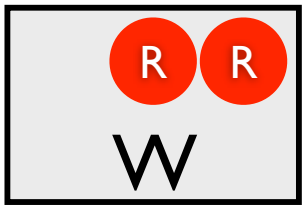
```
0.4 :: heads.
```

```
0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$

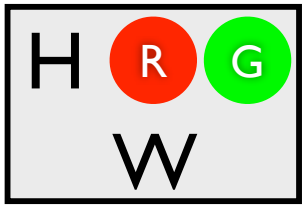


$(1-0.4)$

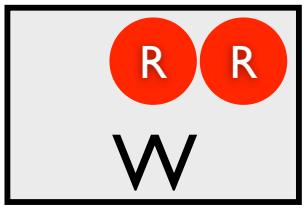
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

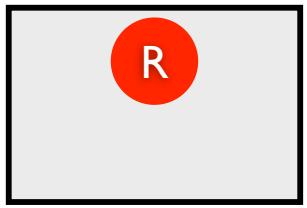
$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$



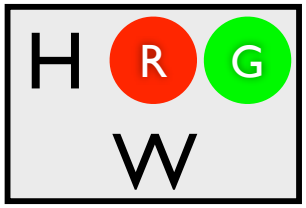
$(1-0.4) \times 0.3$



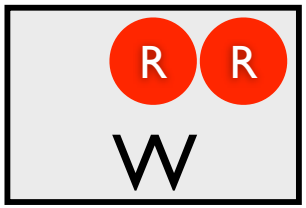
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

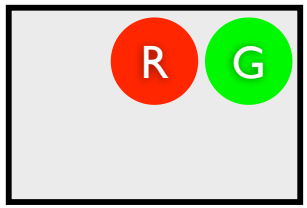
$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$



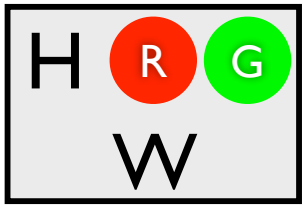
$(1-0.4) \times 0.3 \times 0.3$



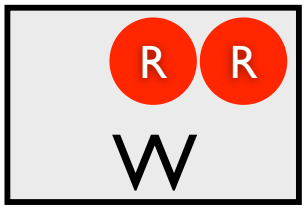
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

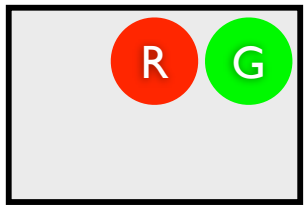
$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$



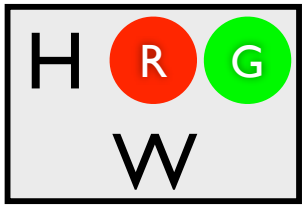
$(1-0.4) \times 0.3 \times 0.3$



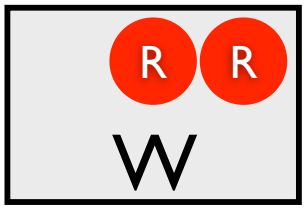
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

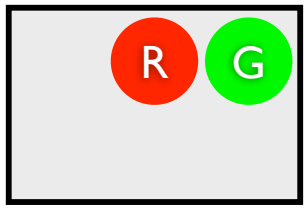
$0.4 \times 0.3 \times 0.3$



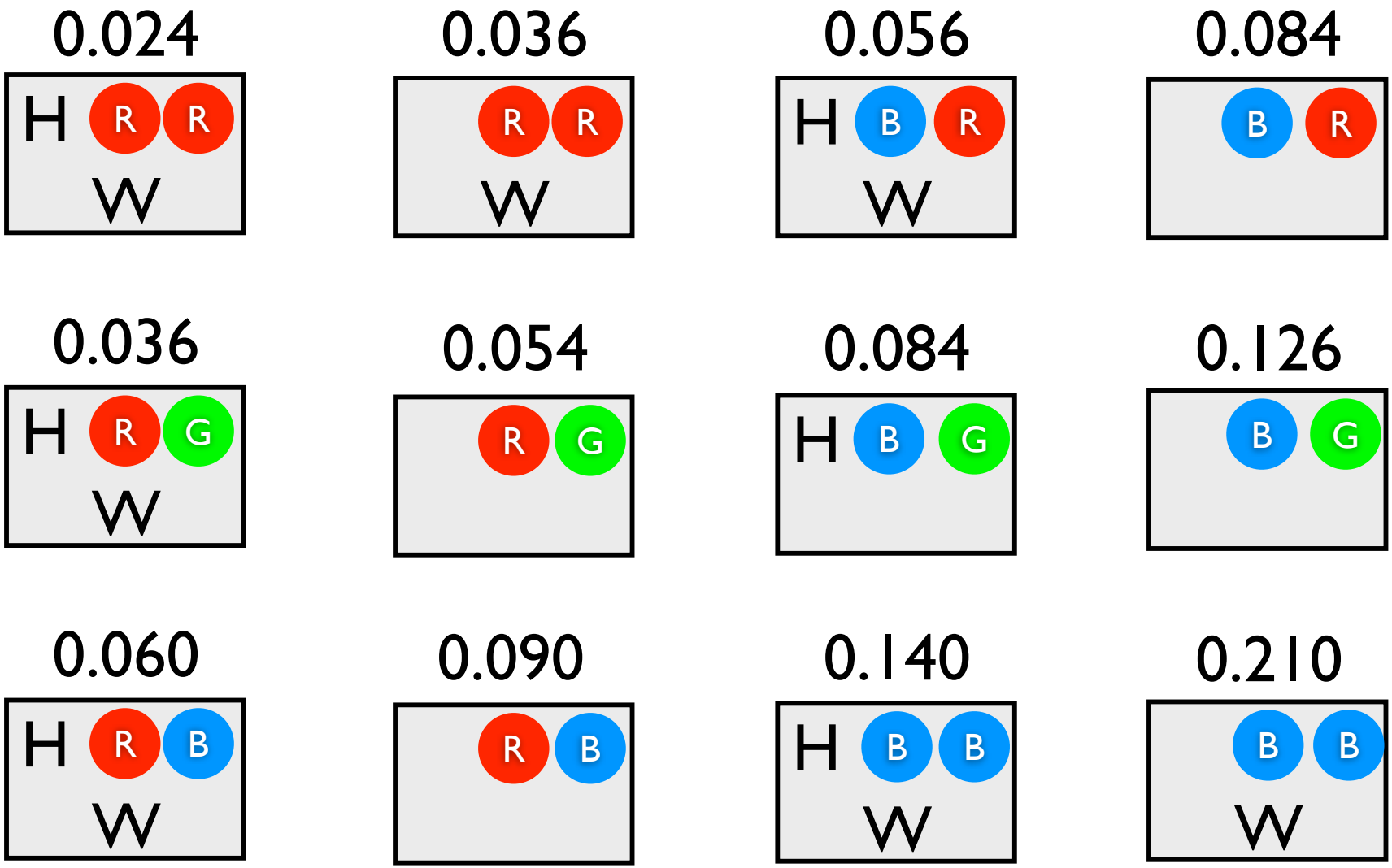
$(1-0.4) \times 0.3 \times 0.2$



$(1-0.4) \times 0.3 \times 0.3$

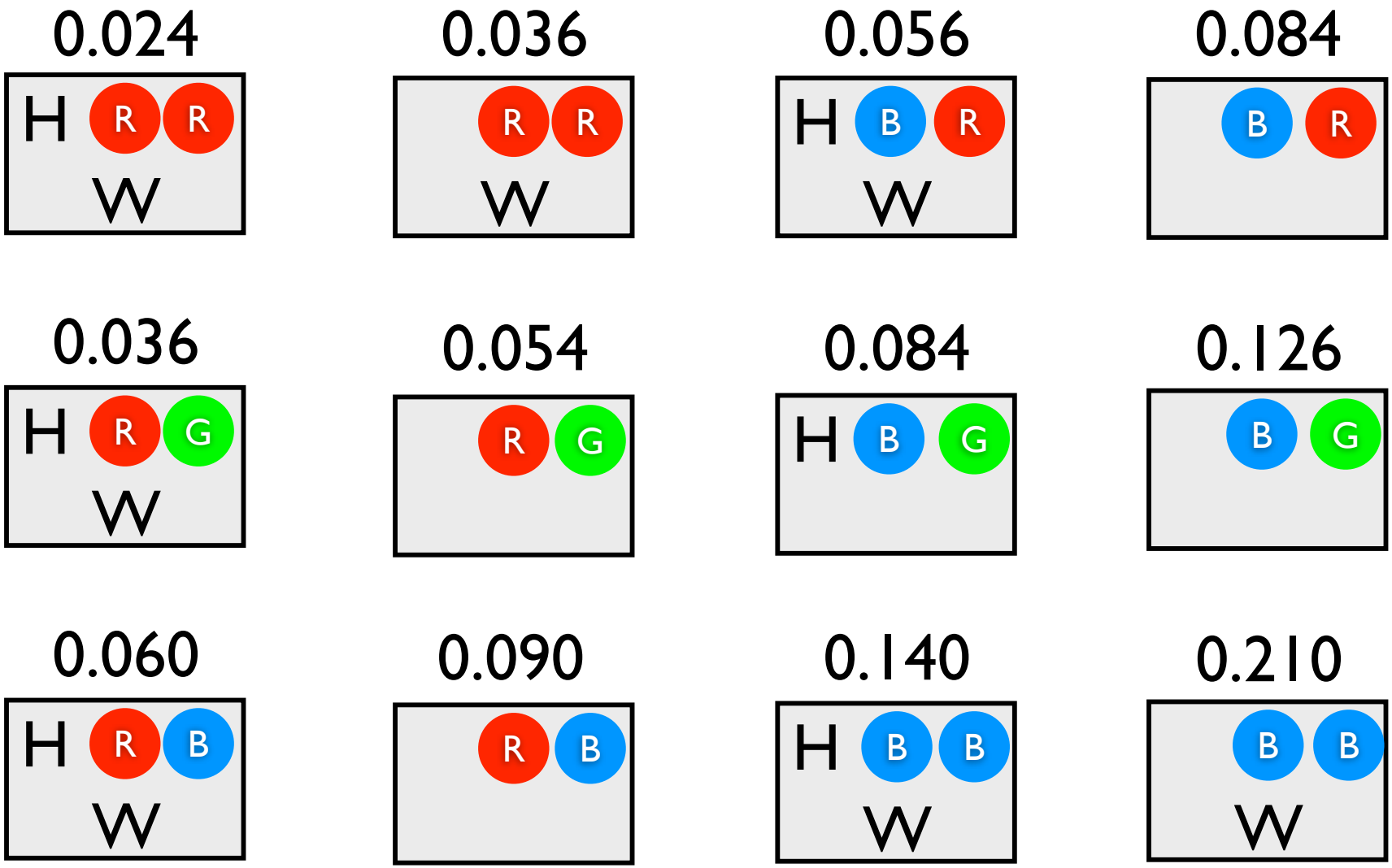


All Possible Worlds



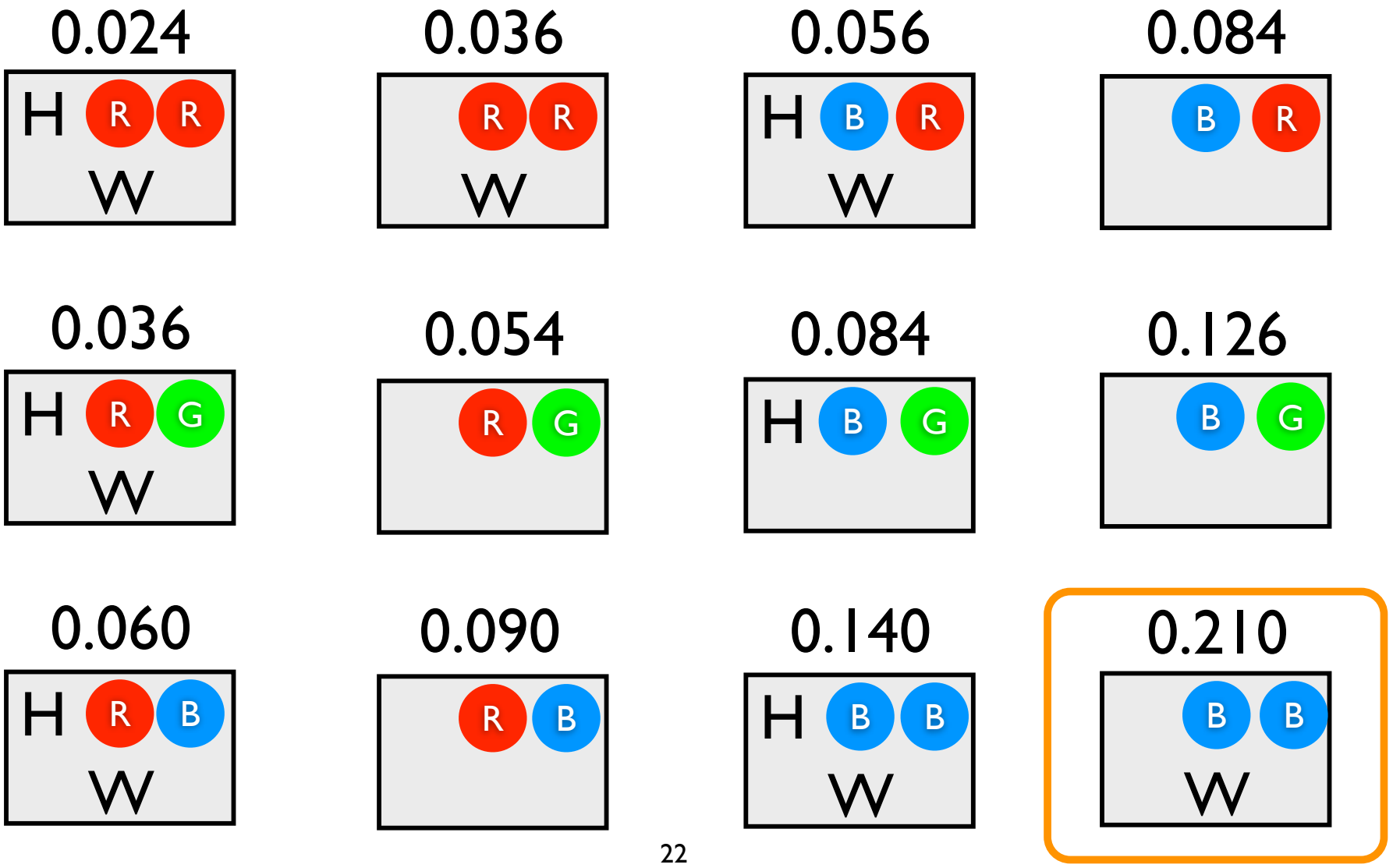
Most likely world where **win** is true?

MPE Inference



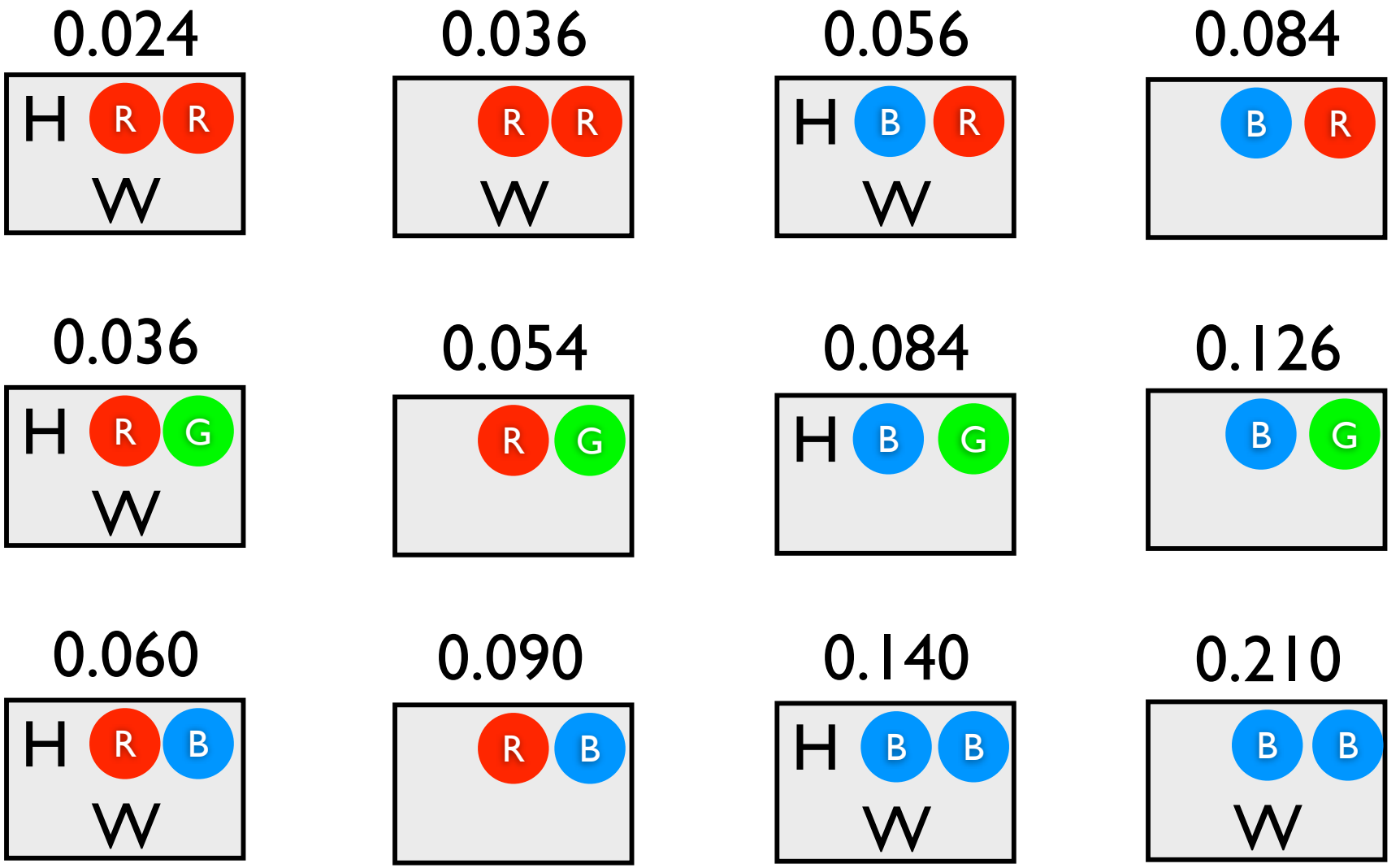
Most likely world where **win** is true?

MPE Inference



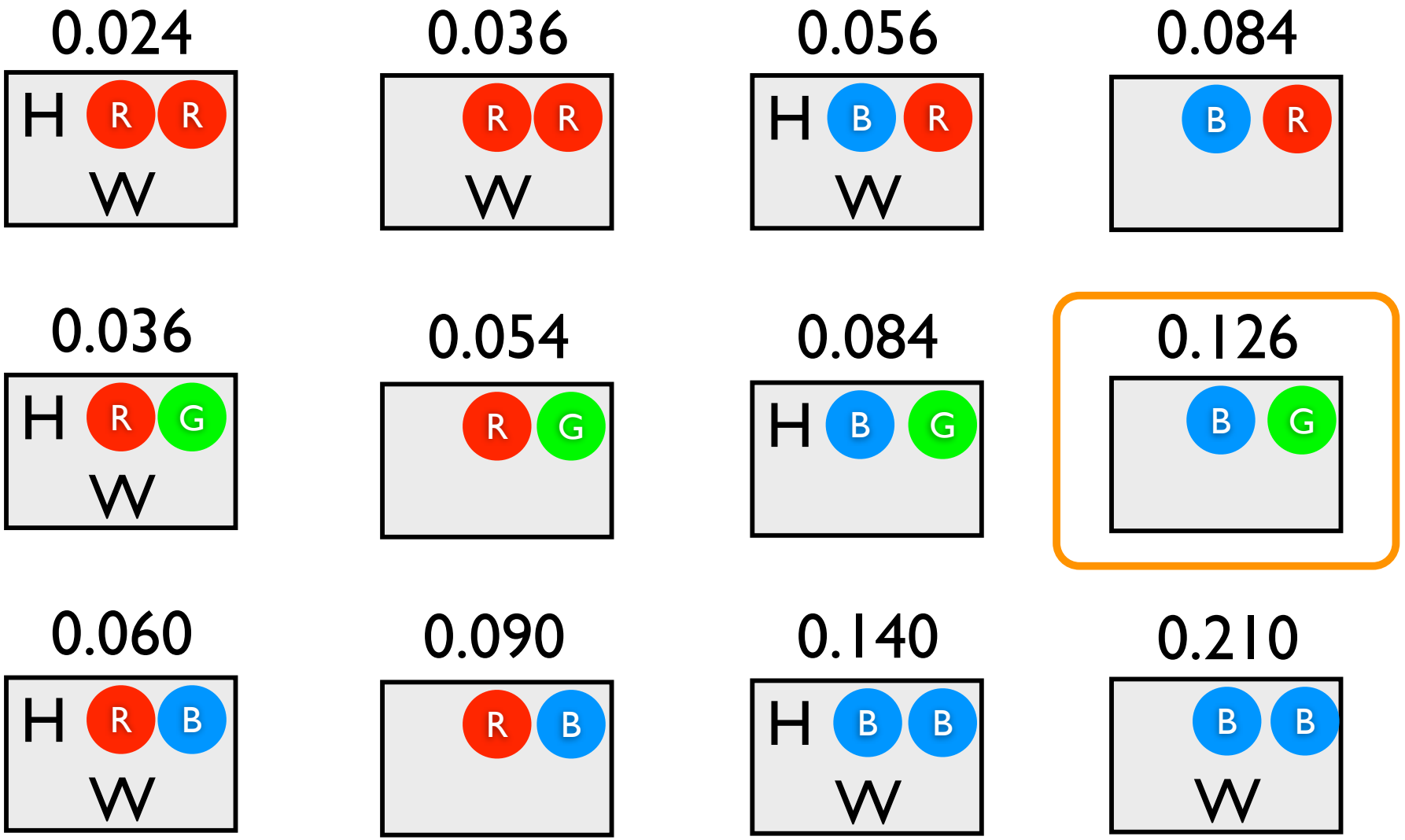
Most likely world where `col(2,blue)` is false?

MPE Inference



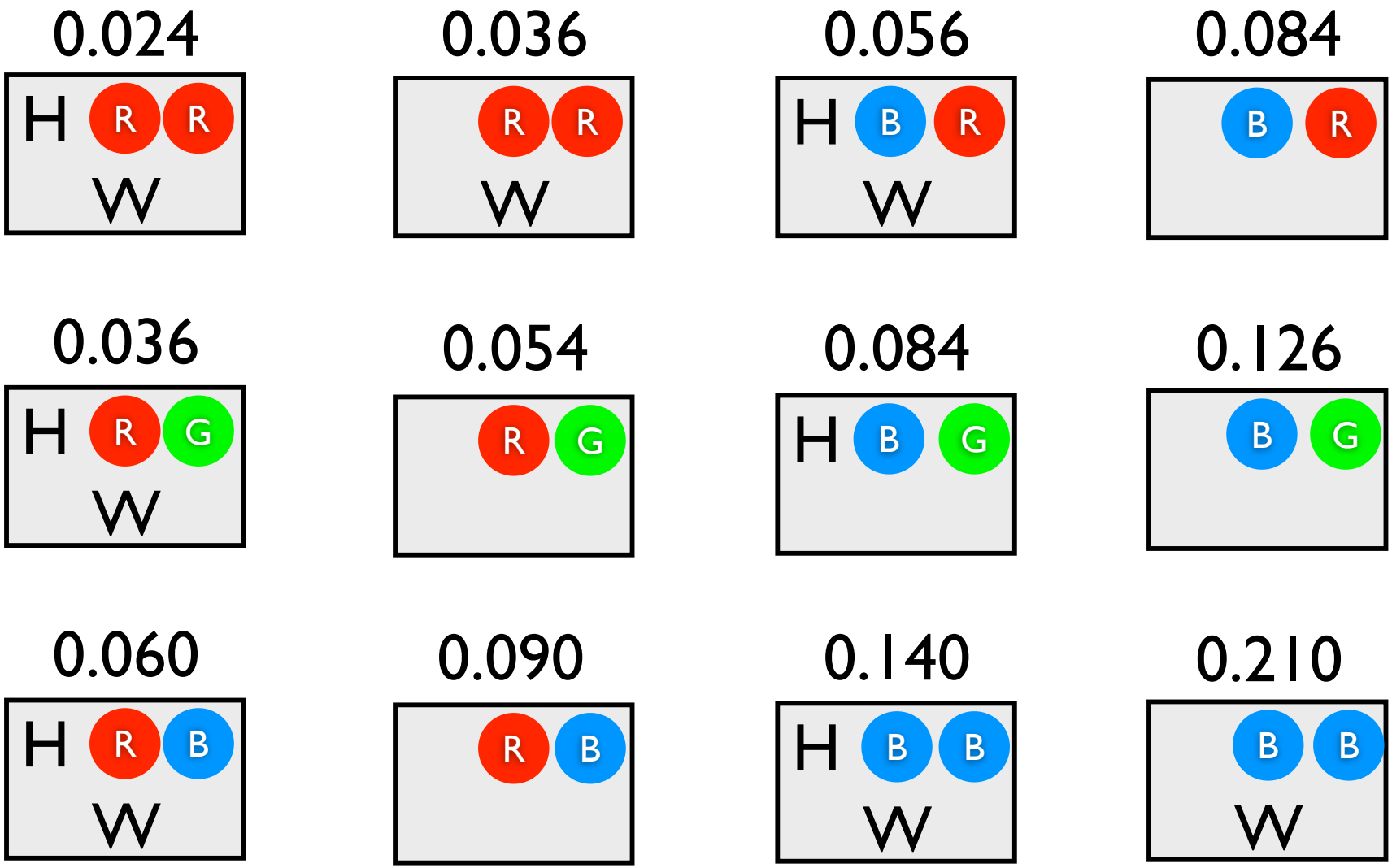
Most likely world where `col(2,blue)` is false?

MPE Inference



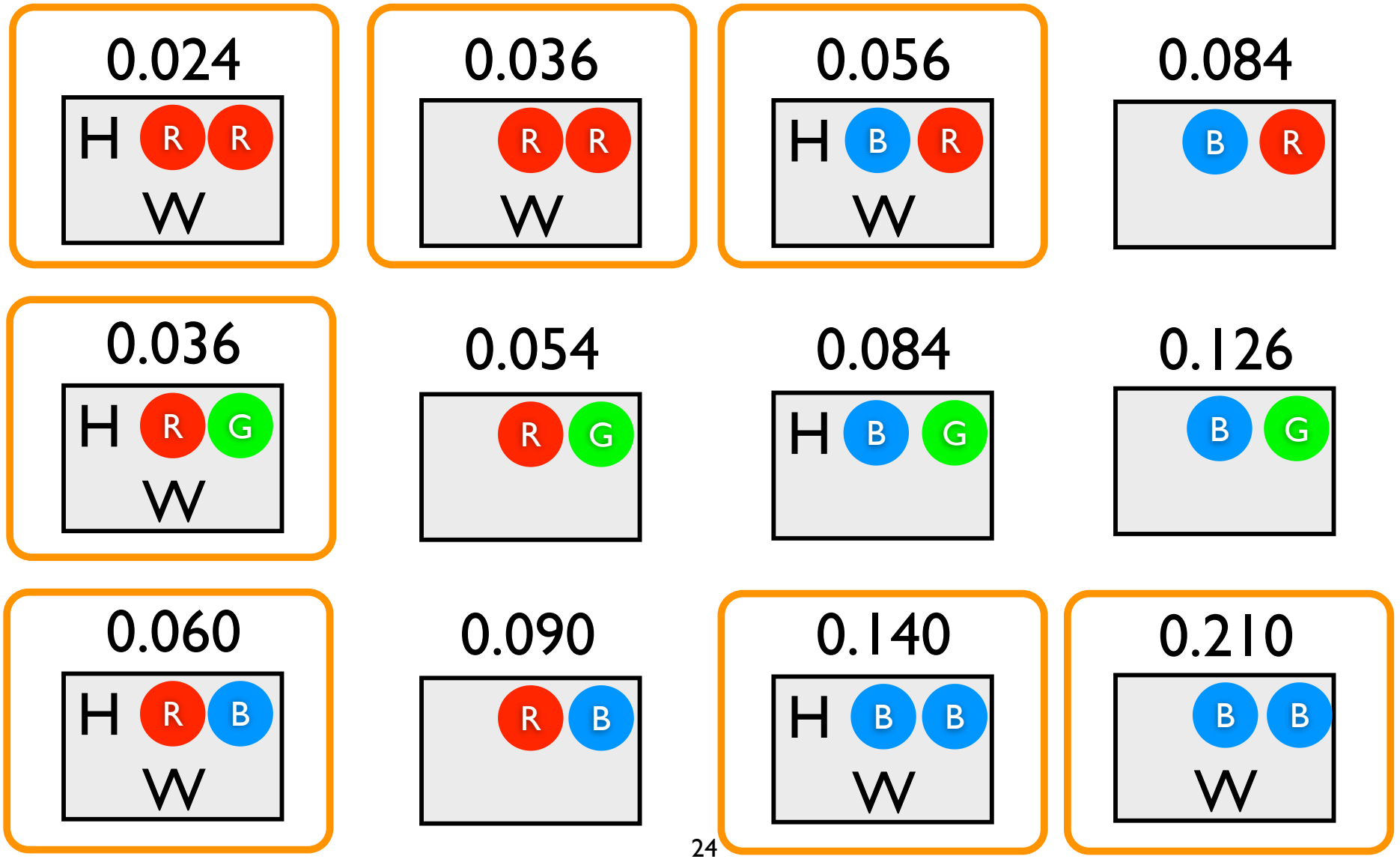
P(win)=?

Marginal Probability



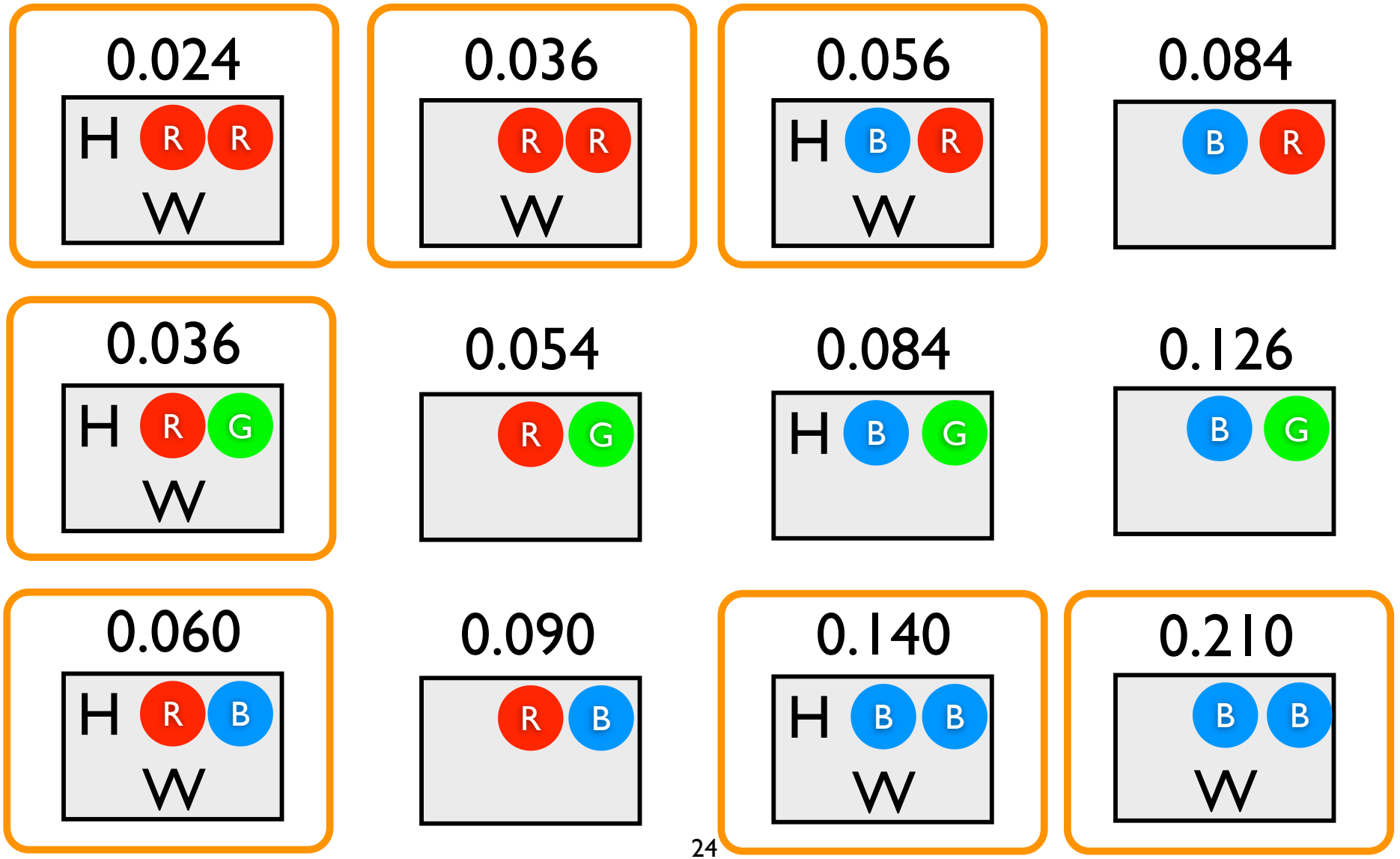
P(win)= Σ

Marginal Probability



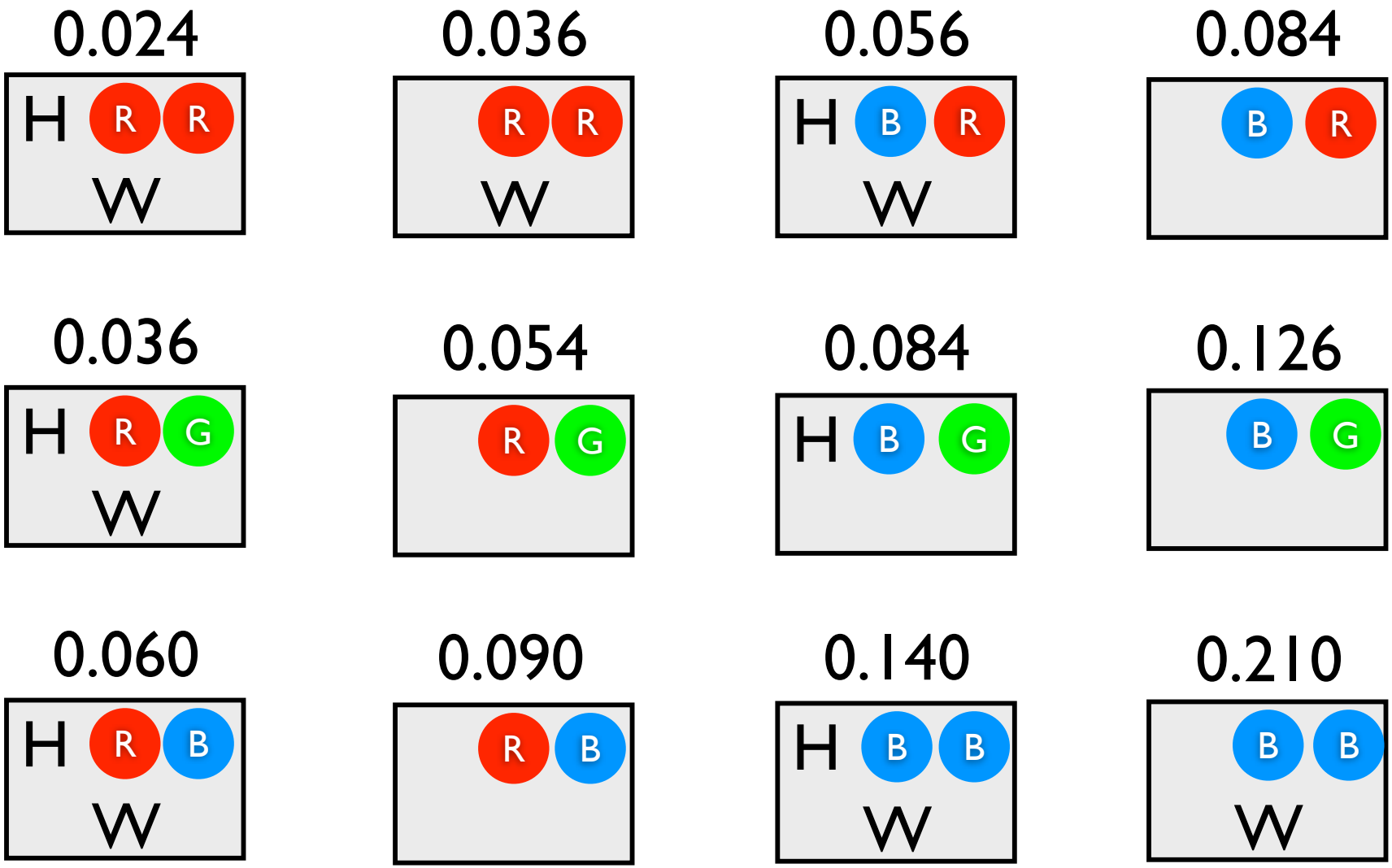
P(win)= Σ =0.562

Marginal Probability



P(win|col(2,green))= ?

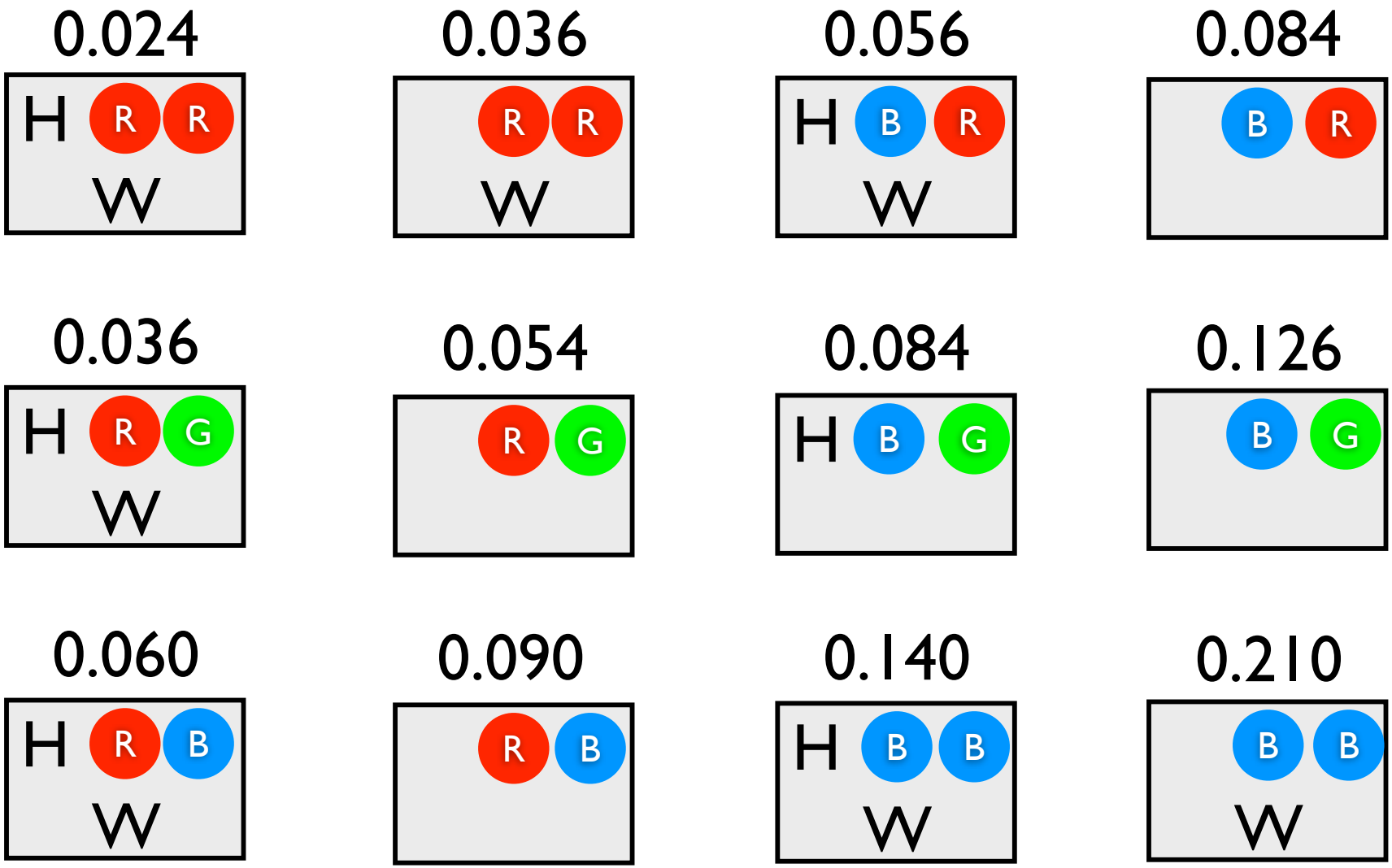
Conditional Probability



$$P(win|col(2,green)) = \Sigma/\Sigma$$

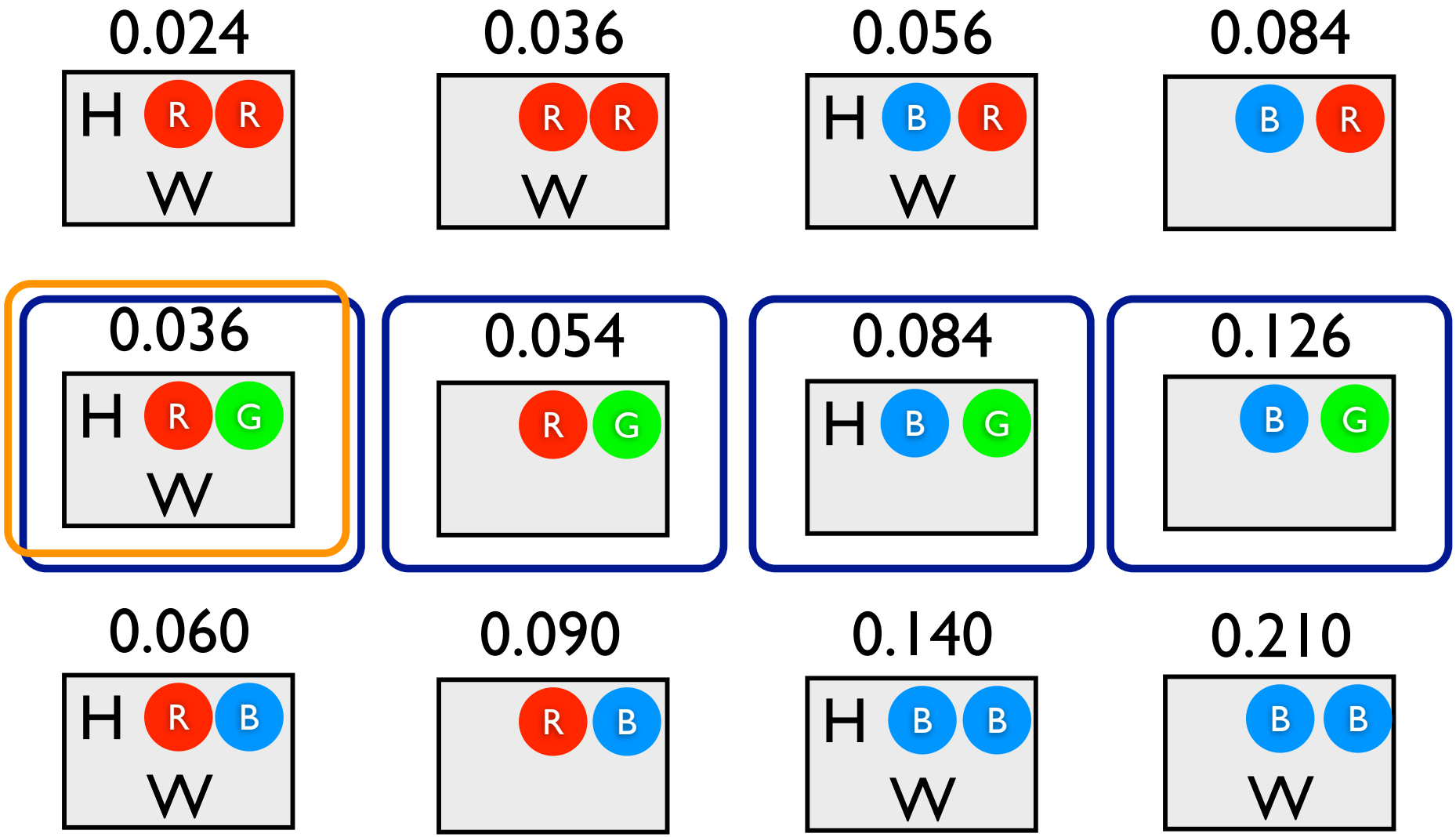
$$= P(win \wedge col(2,green))/P(col(2,green))$$

Conditional Probability



$$P(win|col(2,green)) = \Sigma/\Sigma$$
$$= P(win \wedge col(2,green))/P(col(2,green))$$

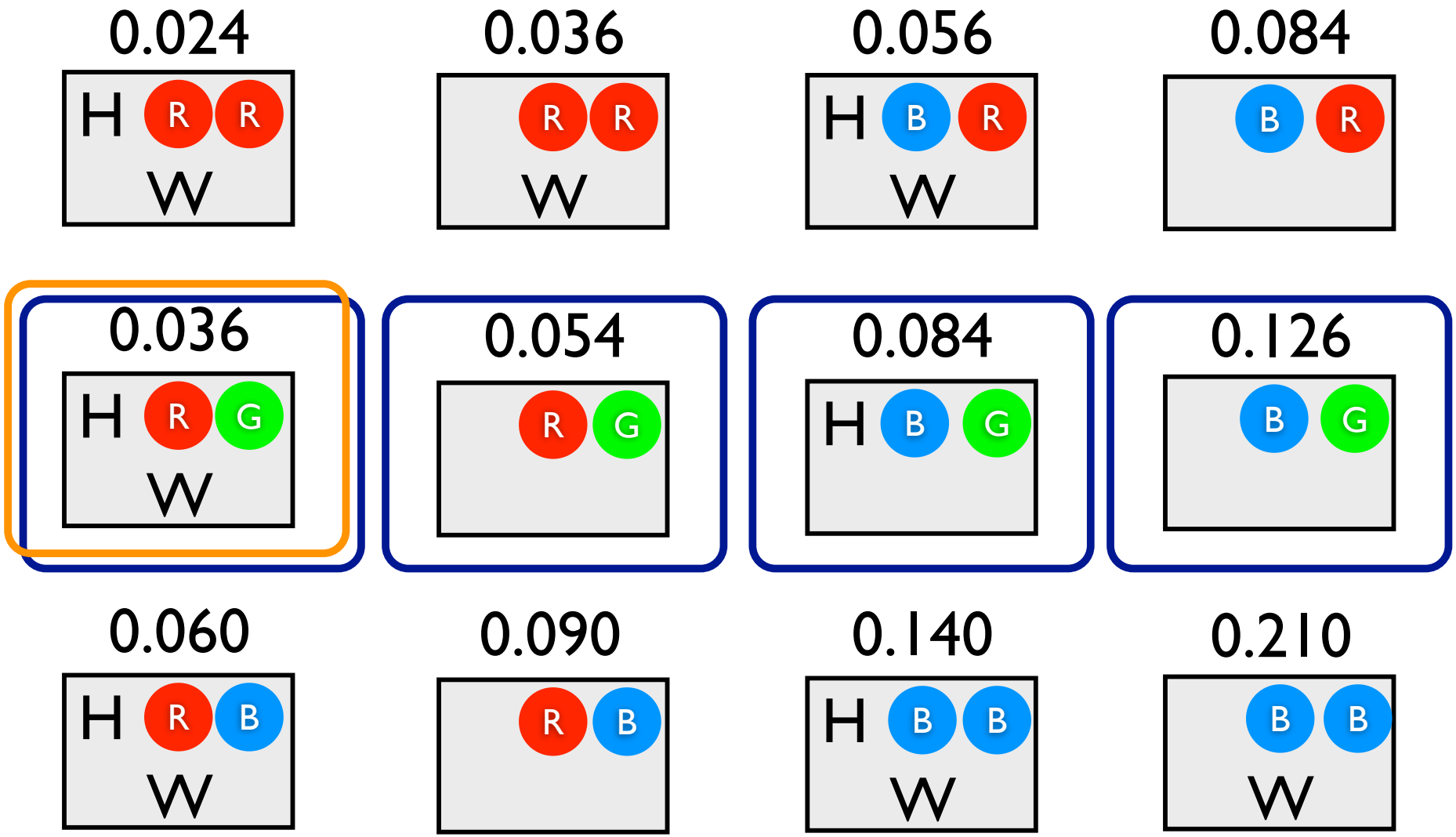
Conditional Probability



$P(win|col(2,green)) = \Sigma/\Sigma$

$= 0.036/0.3 = 0.12$

Conditional Probability



cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```


[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

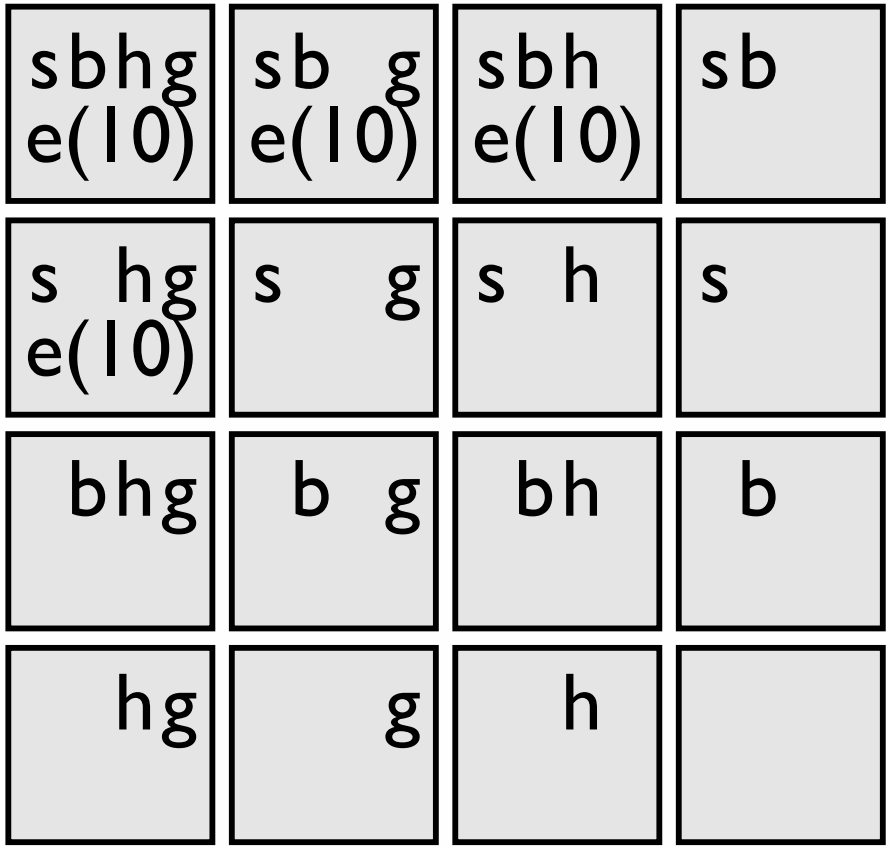
distribution over **all** possible worlds

[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

distribution over **all** possible worlds



[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

constraints

as FOL formulas

treat as evidence

sbhg e(10)	sb g e(10)	sbh e(10)	sb
s hg e(10)	s g	s h	s
bhg	b g	bh	b
hg	g	h	

[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

constraints

as FOL formulas

treat as evidence

	sb g e(10)	sbh e(10)	sb
s hg e(10)	s g	s h	s
bhg	b g	bh	b
hg	g	h	

[Fierens et al, PP 12; Shterionov et al]

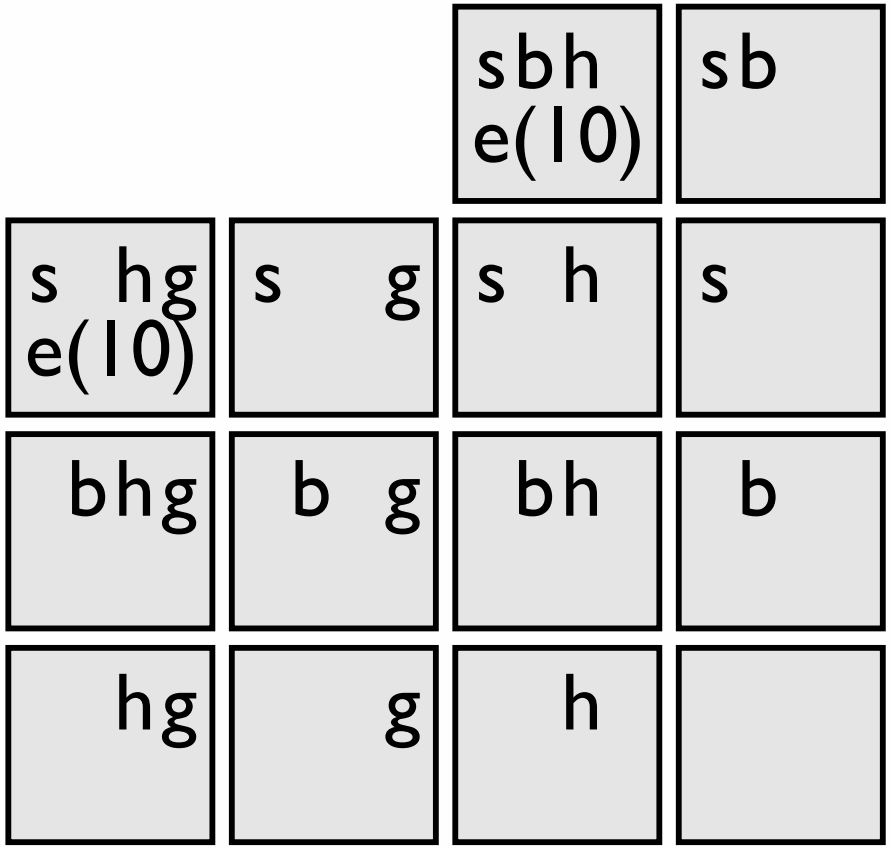
cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

constraints

as FOL formulas

treat as evidence



[Fierens et al, PP 12; Shterionov et al]

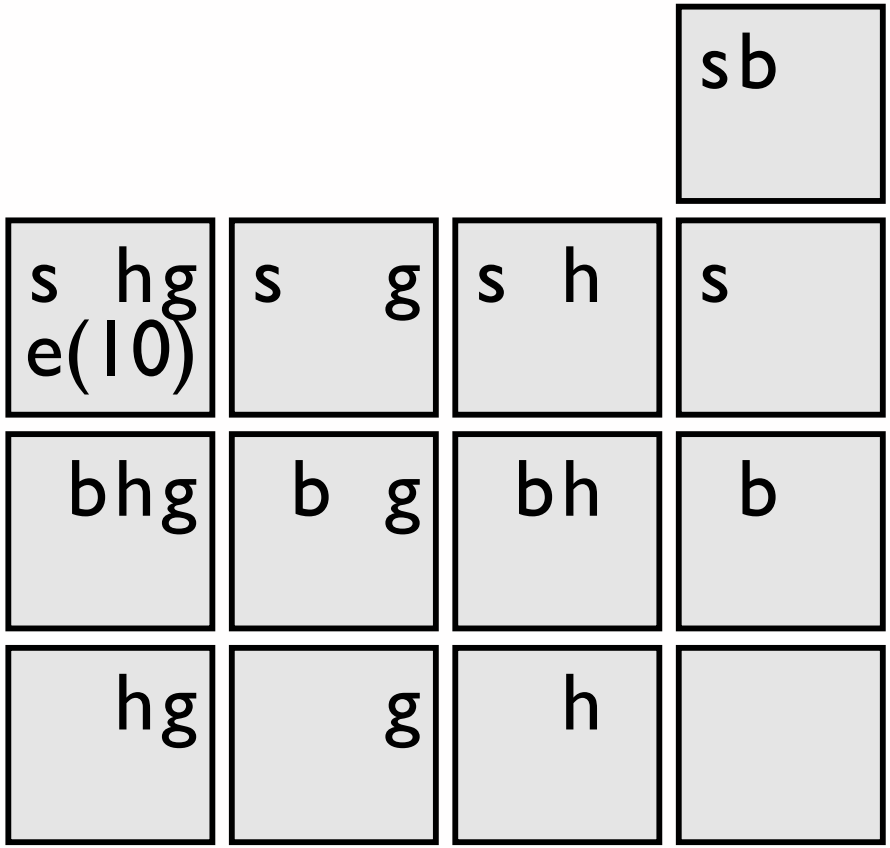
cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

constraints

as FOL formulas

treat as evidence



[Fierens et al, PP 12; Shterionov et al]

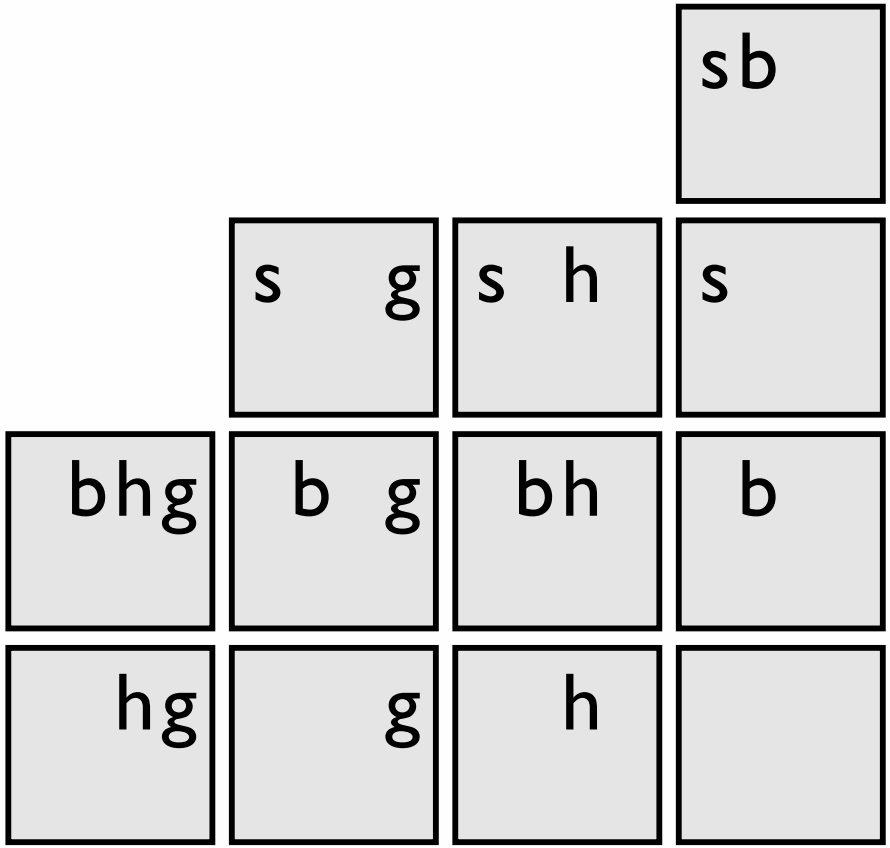
cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

constraints

as FOL formulas

treat as evidence



[Fierens et al, PP 12; Shterionov et al]

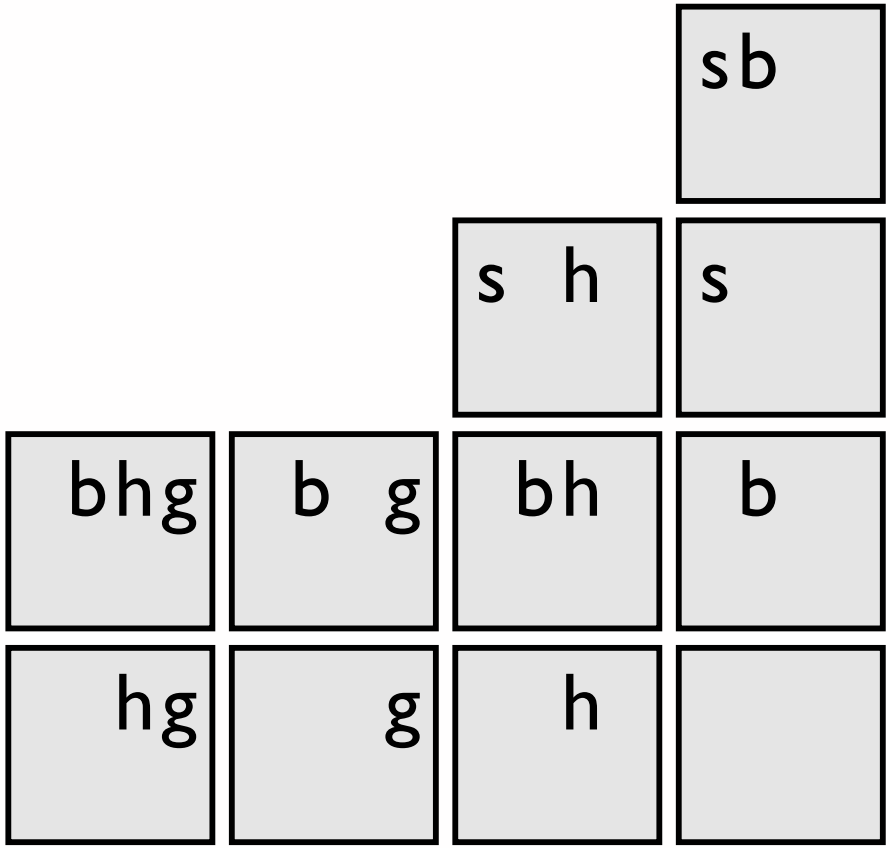
cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

constraints

as FOL formulas

treat as evidence



[Fierens et al, PP 12; Shterionov et al]

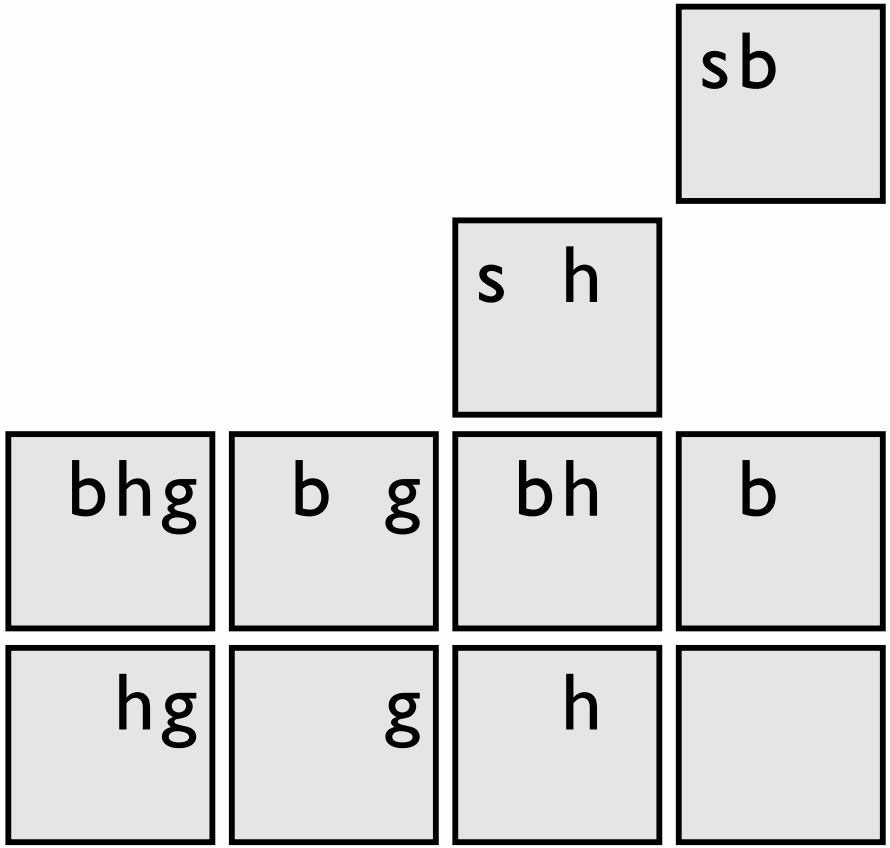
cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

constraints

as FOL formulas

treat as evidence



[Fierens et al, PP 12; Shterionov et al]

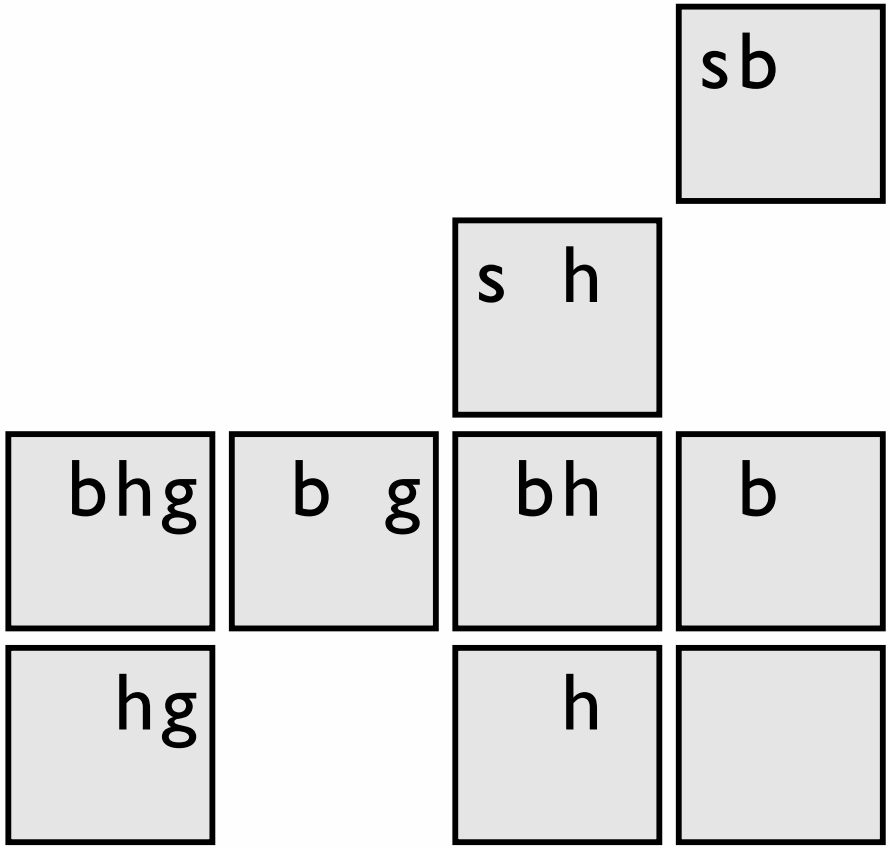
cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

constraints

as FOL formulas

treat as evidence



[Fierens et al, PP 12; Shterionov et al]

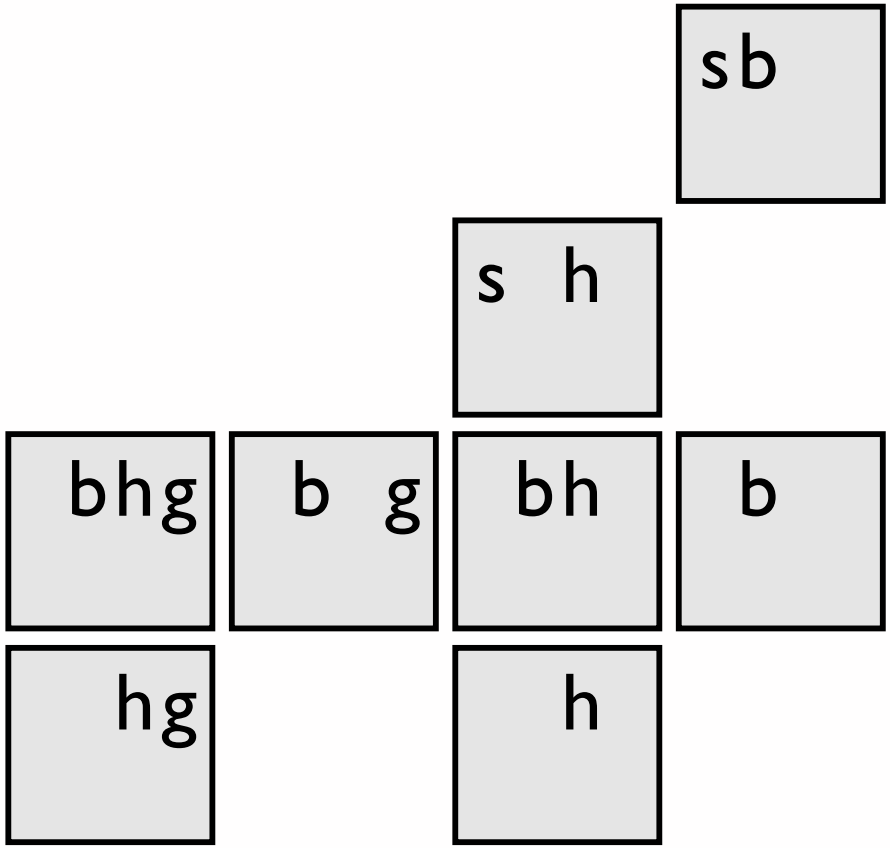
cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

constraints

as FOL formulas

treat as evidence

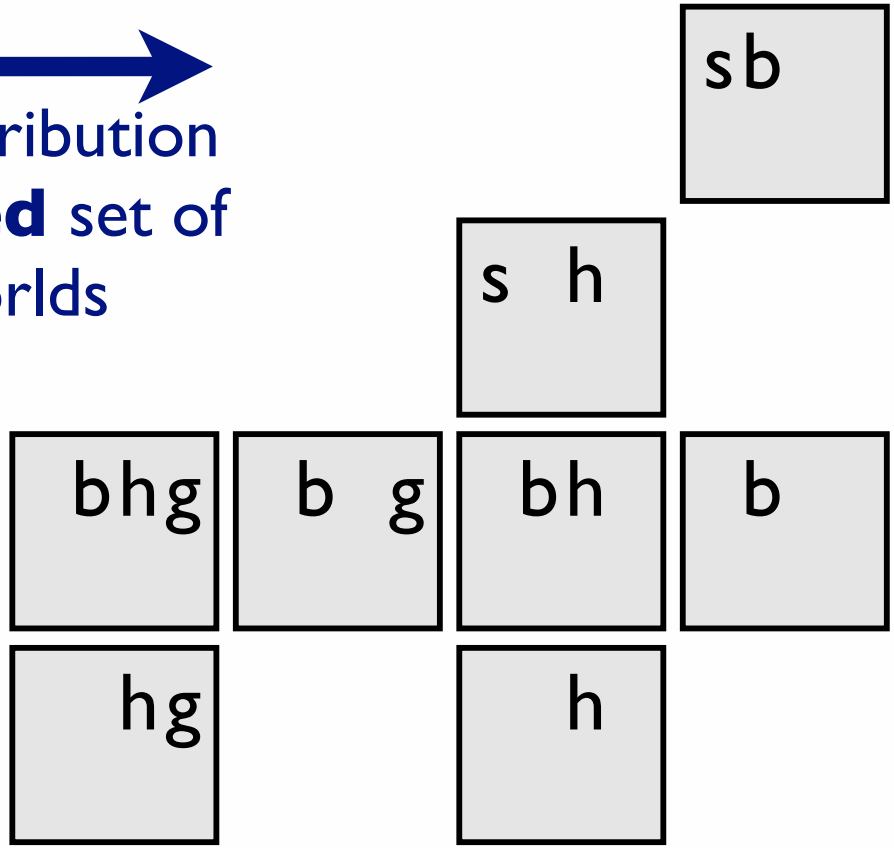


[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

normalized distribution over **restricted** set of possible worlds



constraints
as FOL formulas
treat as evidence

[Fierens et al, PP 12; Shterionov et al]

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

subset of probabilistic facts

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

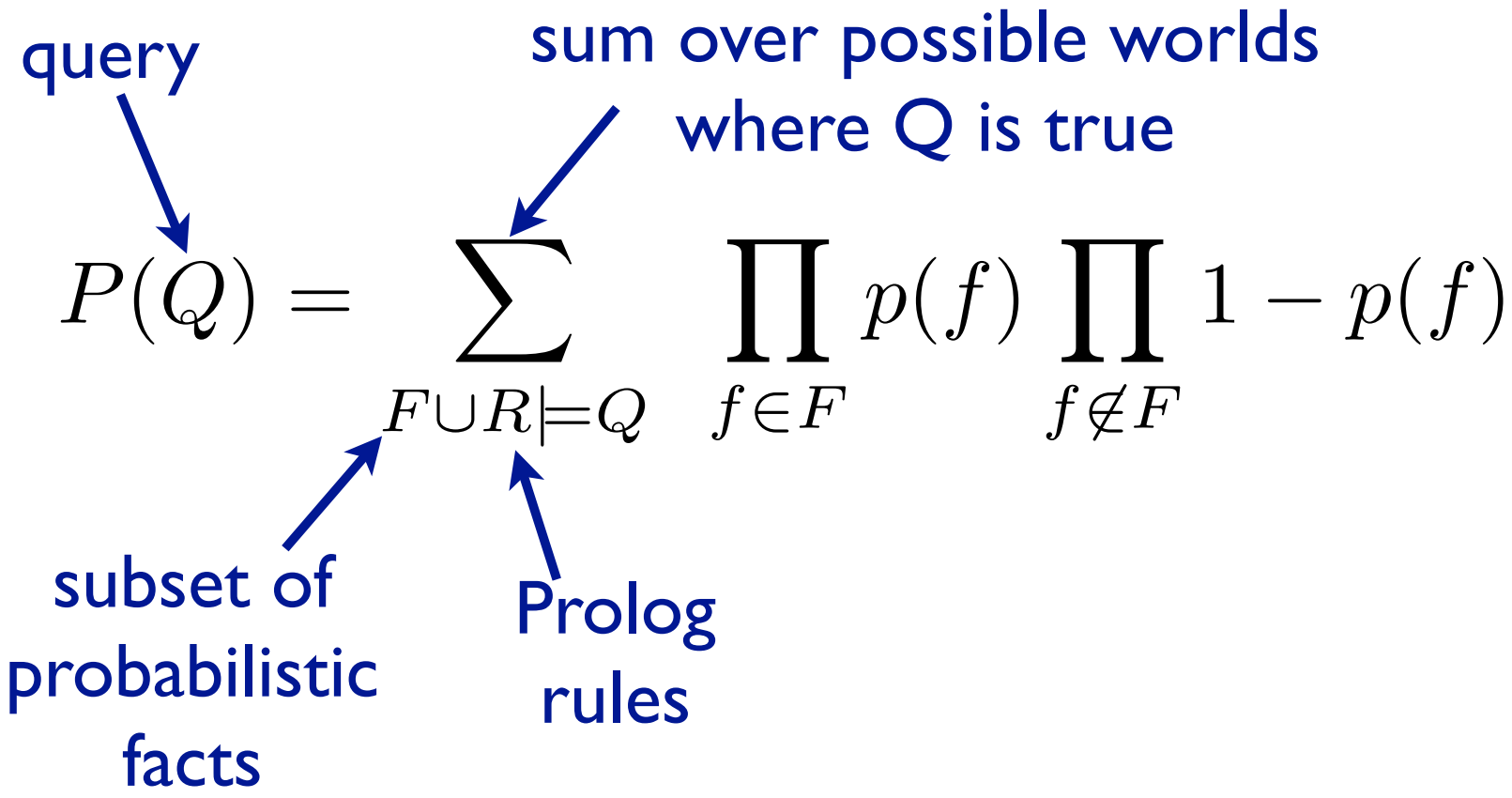
subset of probabilistic facts

Prolog rules

Distribution Semantics

(with probabilistic facts)

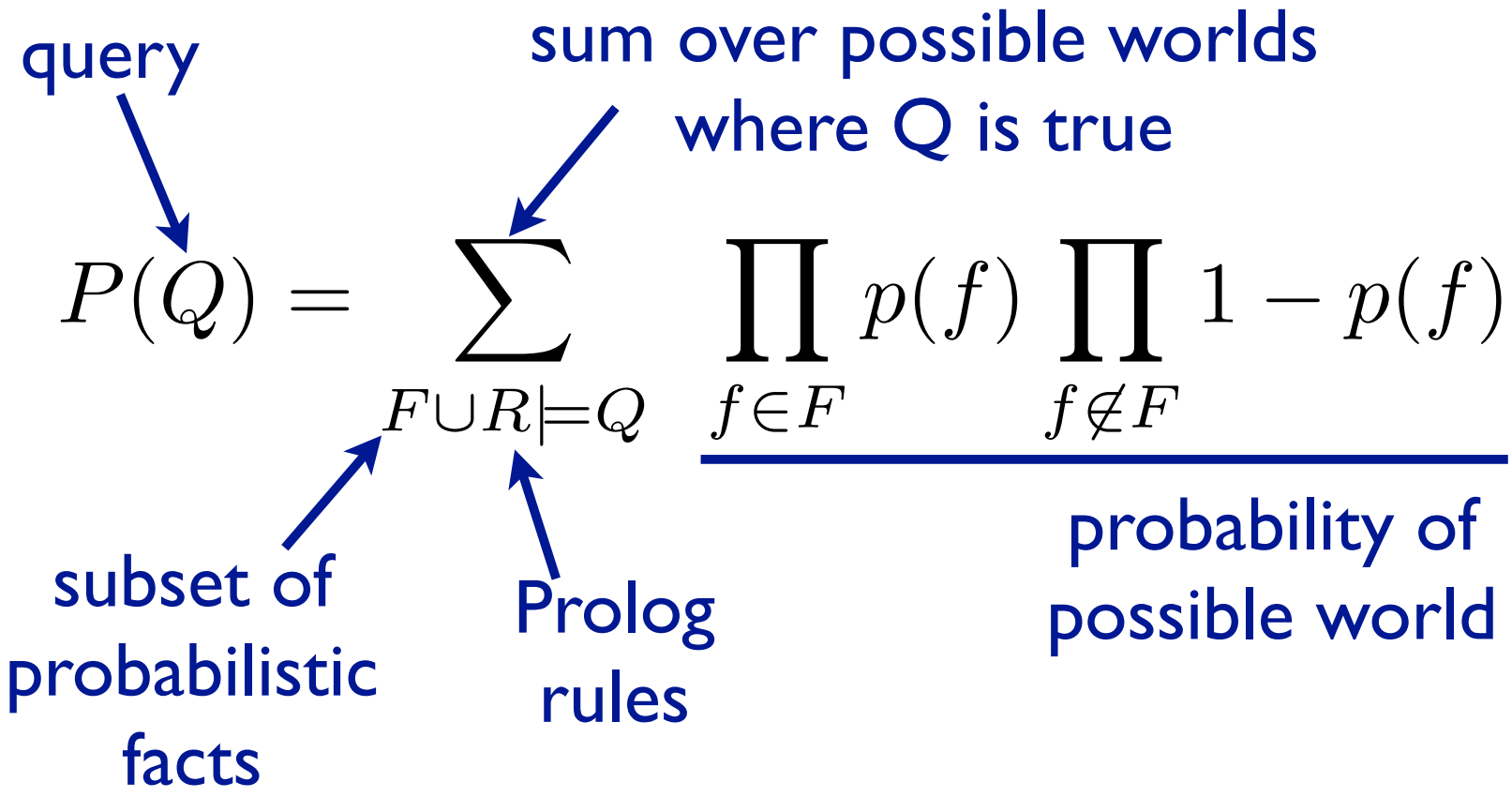
[Sato, ICLP 95]



Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

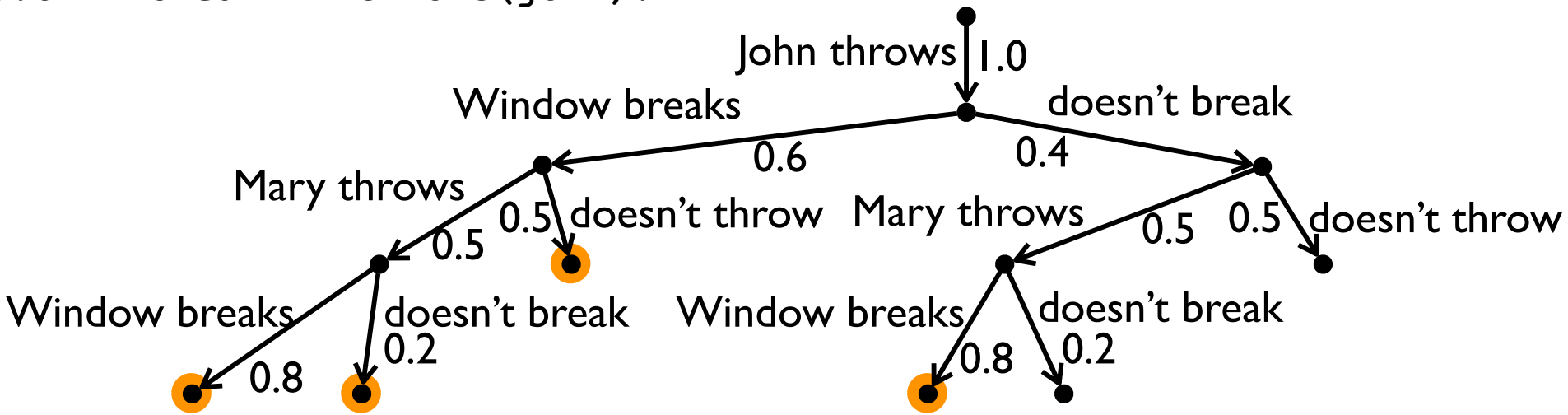


[Vennekens et al, ICLP 04]

Alternative view: CP-Logic

```
throws(john).
0.5::throws(mary).

0.8 :: break <- throws(mary).
0.6 :: break <- throws(john).
```

probabilistic causal laws



P(break)=0.6×0.5×0.8+0.6×0.5×0.2+0.6×0.5+0.4×0.5×0.8

CP-logic [Vennekens et al.]

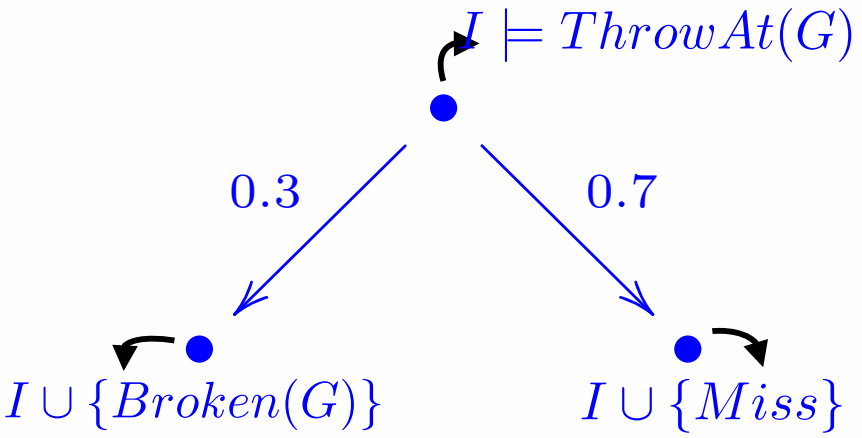
E.g., "throwing a rock at a glass breaks it with probability 0.3 and misses it with probability 0.7"

(Broken(G):0.3) ∨ (Miss 0.7) ← ThrowAt(G).

Note that the actual non-deterministic event ("rock flying at glass") is implicit

Slides CP-logic courtesy Joost Vennekens

Semantics



(Broken(G) 0.3) ∨ (Miss 0.7)
← ThrowAt(G)

Probability tree is an execution model of theory iff:

- Each tree-transition matches causal law
- The tree cannot be extended

Each execution model defines the same probability distribution over final states

Slides CP-logic courtesy Joost Vennekens

Distributional Clauses (DC)

- Discrete- and continuous-valued random variables

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

- Discrete- and continuous-valued random variables

random variable with Gaussian distribution

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
```

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

- Discrete- and continuous-valued random variables

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
stackable(OBot,OTop) :-
     ≃length(OBot) ≥ ≃length(OTop),
     ≃width(OBot) ≥ ≃width(OTop).
```

comparing values of random variables

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

- Discrete- and continuous-valued random variables

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
stackable(OBot,OTop) :-
      ≃length(OBot) ≥ ≃length(OTop),
      ≃width(OBot) ≥ ≃width(OTop).
ontype(Obj,plate) ~ finite([0 : glass, 0.0024 : cup,
                            0 : pitcher, 0.8676 : plate,
                            0.0284 : bowl, 0 : serving,
                            0.1016 : none])
                    :- obj(Obj), on(Obj,O2), type(O2,plate).
```

random variable with discrete distribution

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

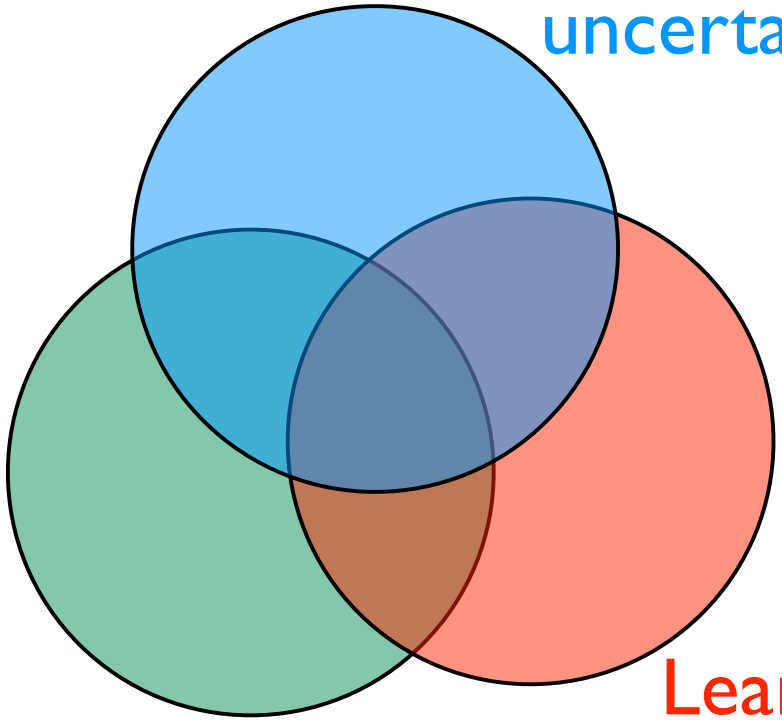
- Discrete- and continuous-valued random variables

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
stackable(OBot,OTop) :-
      ≃length(OBot) ≥ ≃length(OTop),
      ≃width(OBot) ≥ ≃width(OTop).
ontype(Obj,plate) ~ finite([0 : glass, 0.0024 : cup,
                            0 : pitcher, 0.8676 : plate,
                            0.0284 : bowl, 0 : serving,
                            0.1016 : none])
                    :- obj(Obj), on(Obj,O2), type(O2,plate).
```

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

[Suciu et al 2011]

Probabilistic Databases



[Suciu et al 2011]

Probabilistic Databases

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

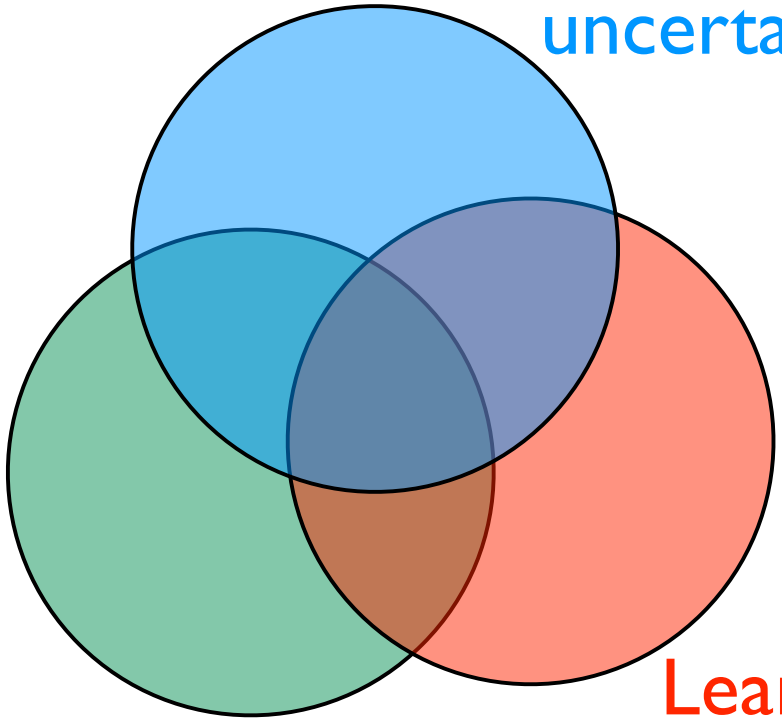
bornIn

person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

relational database



[Suciu et al 2011]

Probabilistic Databases

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

one world

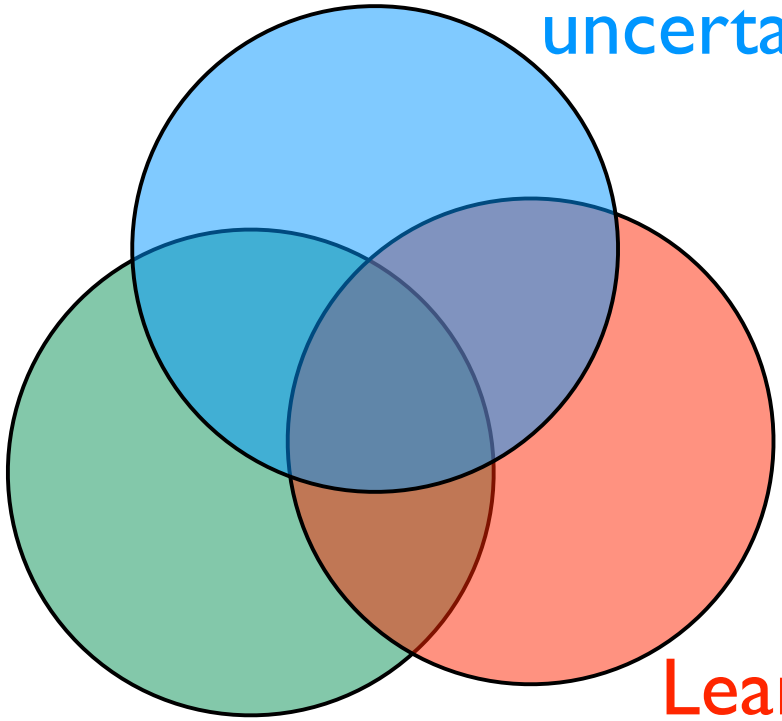
bornIn

person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

relational database



[Suciu et al 2011]

Probabilistic Databases

bornIn

person	city	P
ann	london	0,87
bob	york	0,95
eve	new york	0,90
tom	paris	0,56

cityIn

city	country	P
london	uk	0,99
york	uk	0,75
paris	usa	0,40

tuples as random variables

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

one world

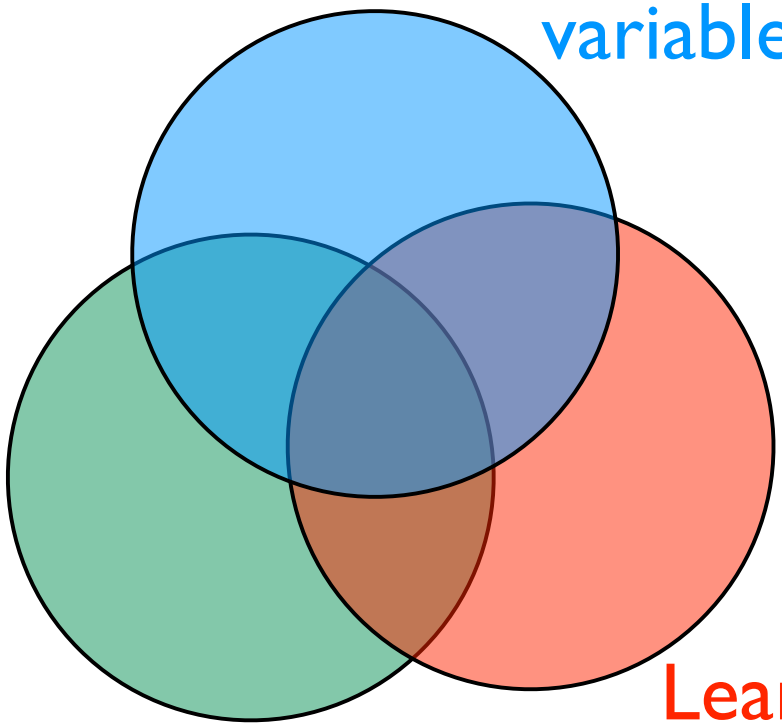
bornIn

person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

relational database



Learning

[Suciu et al 2011]

Probabilistic Databases

several possible worlds

bornIn

person	city	P
ann	london	0,87
bob	york	0,95
eve	new york	0,90
tom	paris	0,56

cityIn

city	country	P
london	uk	0,99
york	uk	0,75
paris	usa	0,40

tuples as random variables

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

one world

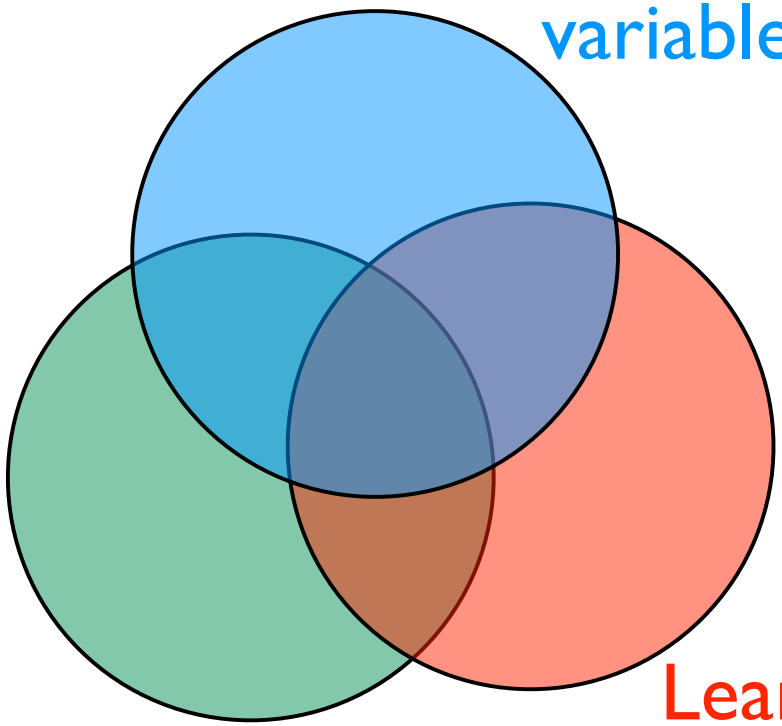
bornIn

person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

relational database



Learning

[Suciu et al 2011]

Probabilistic Databases

several possible worlds

bornIn

person	city	P
ann	london	0,87
bob	york	0,95
		0,90
		0,56

cityIn

city	country	P
london	uk	0,99
york	uk	0,75
paris	usa	0,40

tuples as random variables

probabilistic tables + database queries
→ distribution over possible worlds

```
select
from bornIn x, cityIn y
where x.city=y.city
```

one world

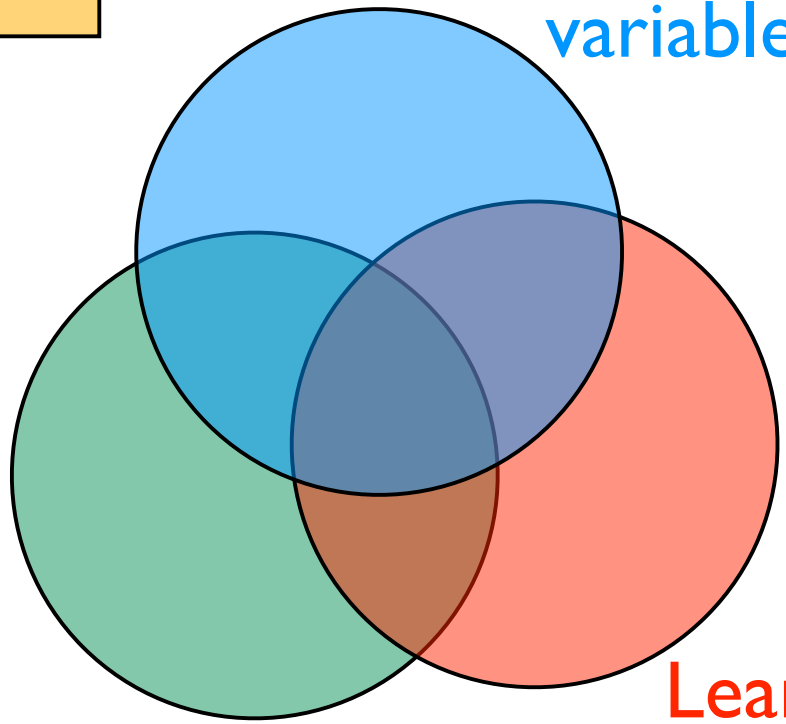
bornIn

person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

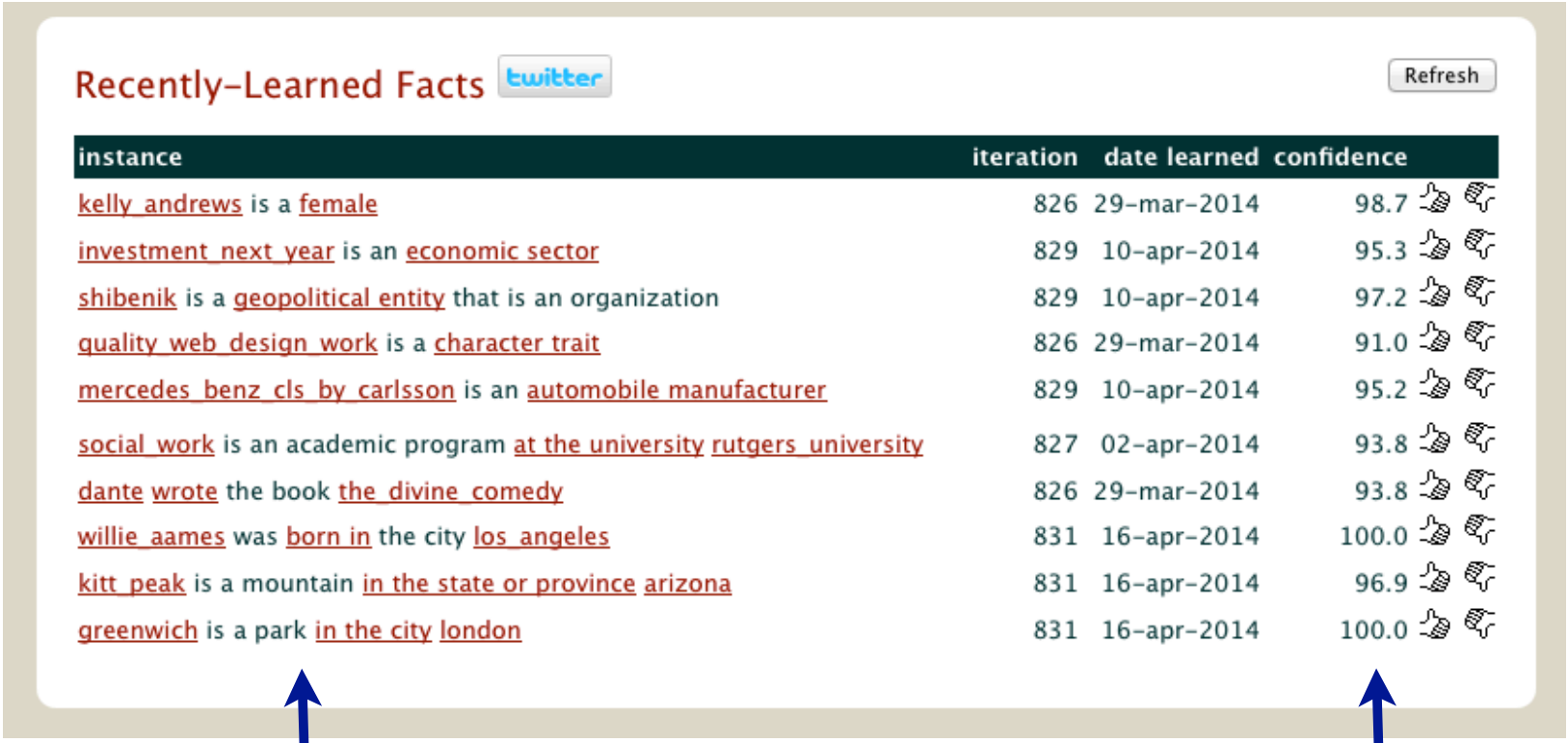
city	country
london	uk
york	uk
paris	usa

relational database



Learning

Example: Information Extraction



Recently-Learned Facts twitter Refresh

instance	iteration	date learned	confidence
kelly_andrews is a female	826	29-mar-2014	98.7
investment_next_year is an economic sector	829	10-apr-2014	95.3
shibenik is a geopolitical entity that is an organization	829	10-apr-2014	97.2
quality_web_design_work is a character trait	826	29-mar-2014	91.0
mercedes_benz_cls_by_carlsson is an automobile manufacturer	829	10-apr-2014	95.2
social_work is an academic program at the university rutgers_university	827	02-apr-2014	93.8
dante wrote the book the_divine_comedy	826	29-mar-2014	93.8
willie_aames was born in the city los_angeles	831	16-apr-2014	100.0
kitt_peak is a mountain in the state or province arizona	831	16-apr-2014	96.9
greenwich is a park in the city london	831	16-apr-2014	100.0

instances for many different relations

degree of certainty

NELL: http://rtw.ml.cmu.edu/rtw/

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

[Example from Suciu et al 2011]

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

same query -
probabilities handled implicitly

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

[Example from Suciu et al 2011]

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

0.96x0.99=0.95

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

[Example from Suciu et al 2011]

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

0.9x0.93=0.83

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

[Example from Suciu et al 2011]

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

0.87x0.93=0.80

[Example from Suciu et al 2011]

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

answer tuples ranked by probability

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

[Example from Suciu et al 2011]

PDB with tuple-level uncertainty in ProbLog?

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

[Example from Suciu et al 2011]

PDB with tuple-level uncertainty in ProbLog?

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

```
0.96::producesProduct(sony,walkman).
0.96::producesProduct(microsoft,mac_os_x).
0.96::producesProduct(ibm,personal_computer).
0.9::producesProduct(microsoft,mac_os).
0.9::producesProduct(adobe,adobe_indesign).
0.87::producesProduct(adobe,adobe_dreamweaver).
...
```


[Example from Suciu et al 2011]

PDB with tuple-level uncertainty in ProbLog?

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and y.City='san_jose'
```

PDB with tuple-level uncertainty in ProbLog?

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and y.City='san_jose'
```

```
result(Product,Company) :-
    producesProduct(Company,Product),
    headquarteredIn(Company,san_jose).
query(result(_,_)).
```

PDB with tuple-level uncertainty in ProbLog?

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and y.City='san_jose'
```

```
result(Product,Company) :-
    producesProduct(Company,Product),
    headquarteredIn(Company,san_jose).
query(result(_,_)).
```

PDB with attribute-level uncertainty in ProbLog?

color

item	color	P
mug	green blue	0.65 0.35
plate	pink red purple	0.23 0.14 0.63

PDB with attribute-level uncertainty in ProbLog?

color

item	color	P
mug	green blue	0.65 0.35
plate	pink red purple	0.23 0.14 0.63

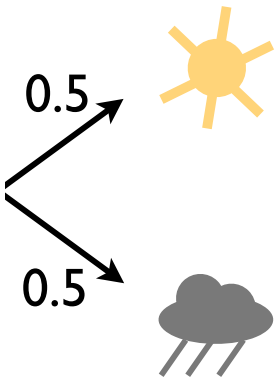
```
0.65::color(mug,green); 0.35::color(mug,blue) <- true.
0.23::color(plate,pink); 0.14::color(plate,red);
                                0.63::color(plate,purple) <- true.
```

ProbLog by example:

Rain or sun?

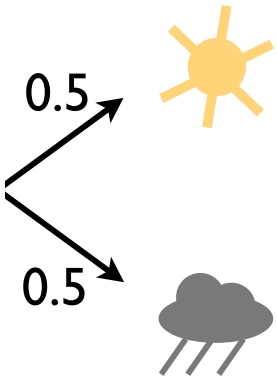
ProbLog by example:

Rain or sun?



ProbLog by example:

Rain or sun?

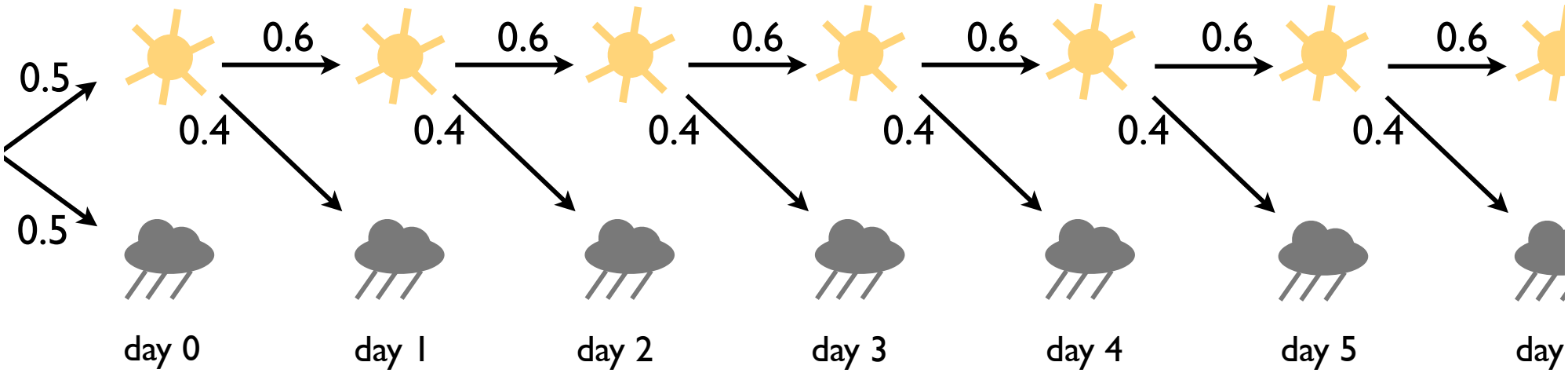


day 0

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

ProbLog by example:

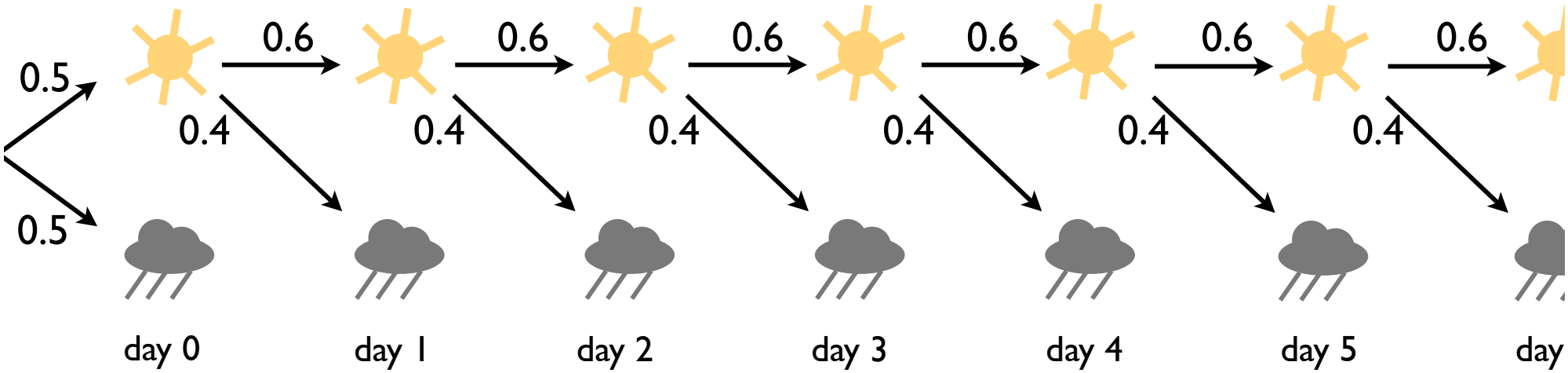
Rain or sun?




```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

ProbLog by example:

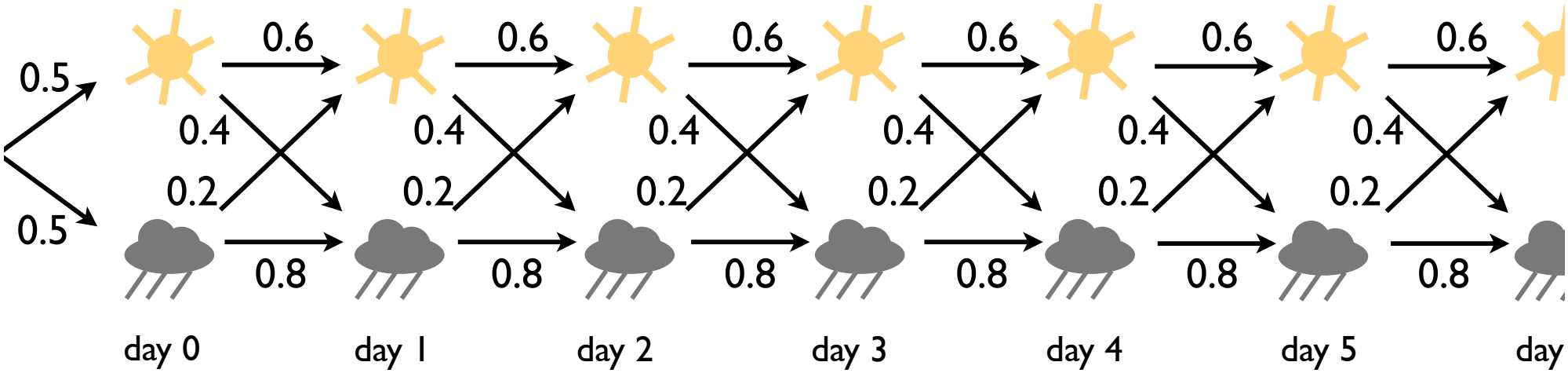
Rain or sun?




```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                    <- T>0, Tprev is T-1, weather(sun,Tprev).
```

ProbLog by example:

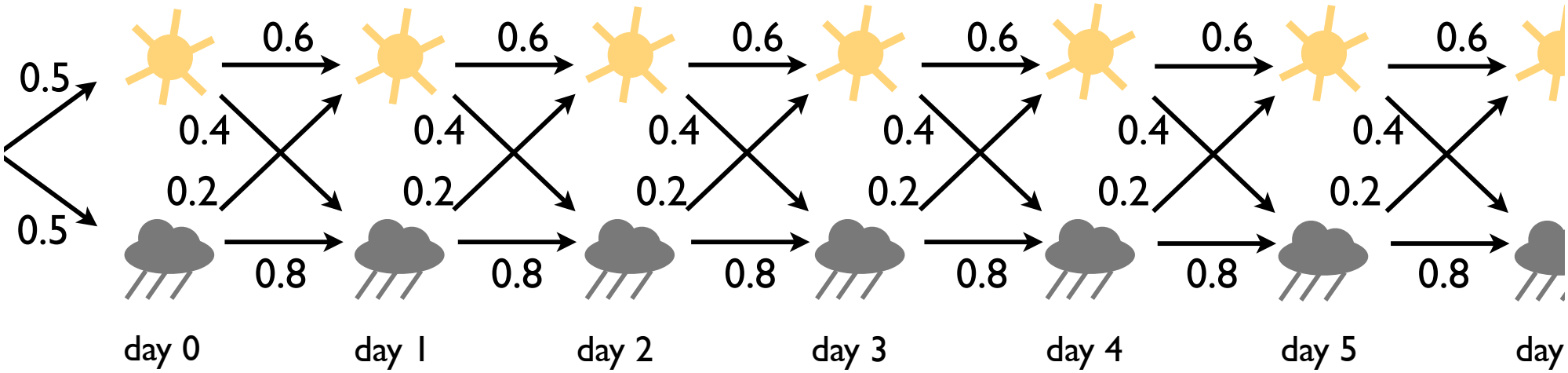
Rain or sun?




```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                    <- T>0, Tprev is T-1, weather(sun,Tprev).
```

ProbLog by example:

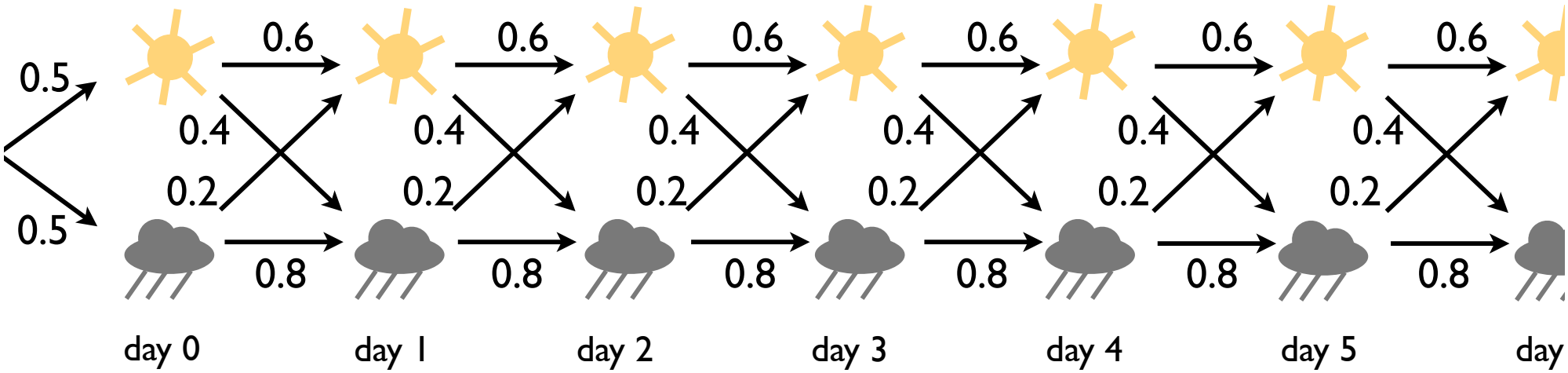
Rain or sun?




```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                  <- T>0, Tprev is T-1, weather(sun,Tprev).
0.2::weather(sun,T) ; 0.8::weather(rain,T)
                  <- T>0, Tprev is T-1, weather(rain,Tprev).
```

ProbLog by example:

Rain or sun?



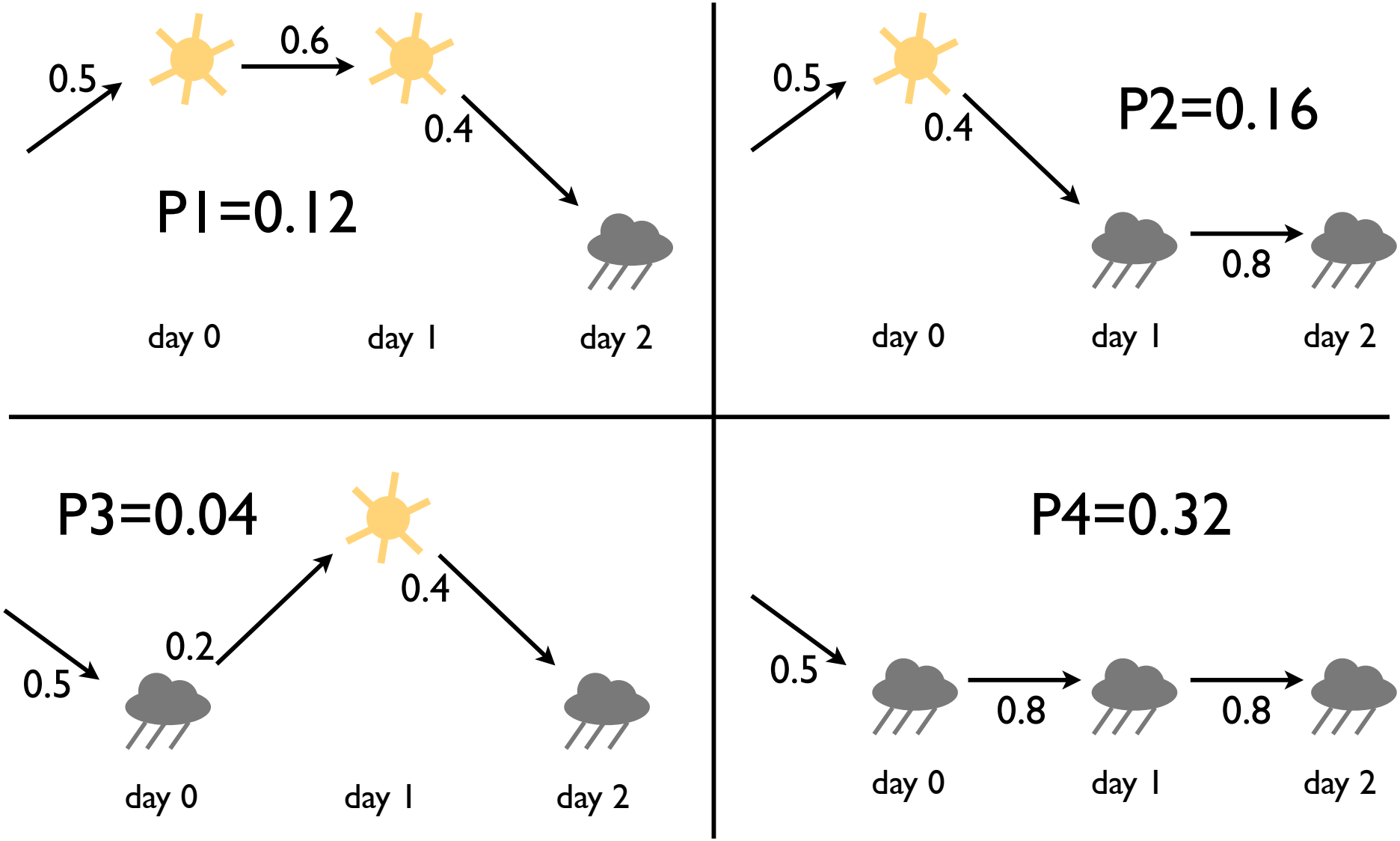

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                  <- T>0, Tprev is T-1, weather(sun,Tprev).
0.2::weather(sun,T) ; 0.8::weather(rain,T)
                  <- T>0, Tprev is T-1, weather(rain,Tprev).
```

infinite possible worlds! BUT: finitely many partial worlds suffice to answer any given ground query

Possible worlds

```
?- weather(rain,2).
```

P=P1+P2+P3+P4



ProbLog

- **probabilistic choices** + their **consequences**
- probability distribution over **possible worlds**
- how to efficiently answer **questions**?
  - most probable world (MPE inference)
  - probability of query (computing marginals)
  - probability of query given evidence

Summary: ProbLog Syntax

- input database: ground facts

```
person(bob).
```

- probabilistic facts

```
0.5::stress(bob).
```

- typed probabilistic facts (body deterministic)

```
0.5::stress(X) :- person(X).
```

- flexible probabilities

```
P::pack(Item) :- weight(Item,W),
                 P is 1.0/W.
```

- annotated disjunctions

```
0.4::asthma(X) <- smokes(X).
```

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.
```

- Prolog clauses

```
smokes(X) :- influences(Y,X), smokes(Y).
```

```
excess([I|R],Limit) :- \+pack(I), excess(R,Limit).
```

Mutually Exclusive Rules:

no two rules apply simultaneously

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                <- T>0, Tprev is T-1, weather(sun,Tprev).

0.2::weather(sun,T) ; 0.8::weather(rain,T)
                <- T>0, Tprev is T-1, weather(rain,Tprev).
```

Mutually Exclusive Rules:

no two rules apply simultaneously

first rule for day 0, others for later days

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                <- T>0, Tprev is T-1, weather(sun,Tprev).

0.2::weather(sun,T) ; 0.8::weather(rain,T)
                <- T>0, Tprev is T-1, weather(rain,Tprev).
```

Mutually Exclusive Rules:

no two rules apply simultaneously

first rule for day 0, others for later days

day 0: either sun or rain

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                <- T>0, Tprev is T-1, weather(sun,Tprev).

0.2::weather(sun,T) ; 0.8::weather(rain,T)
                <- T>0, Tprev is T-1, weather(rain,Tprev).
```

Mutually Exclusive Rules:

no two rules apply simultaneously

first rule for day 0, others for later days

day 0: either sun or rain

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                <- T>0, Tprev is T-1, weather(sun,Tprev).

0.2::weather(sun,T) ; 0.8::weather(rain,T)
                <- T>0, Tprev is T-1, weather(rain,Tprev).
```

rules for T>0 cover mutually exclusive cases on previous day

[Sato and Kameya, JAIR 01]

PRISM

- Another probabilistic Prolog based on the distribution semantics
- Mutual exclusiveness assumption
  - allows for efficient inference by dynamic programming, cf. probabilistic grammars
  - but excludes certain models, e.g., smokers
- http://sato-www.cs.titech.ac.jp/prism/

PRISM

- "multi-valued random switches" = annotated disjunctions with body *true*
- switch gives fresh result on each call
- Prolog rules
- limited support for negation (compiling away)

stochastic memoization

Weather in PRISM

```
values(init,[sun,rain]).
values(tr(_),[sun,rain]).

:- set_sw(init,[0.5,0.5]).
:- set_sw(tr(sun),[0.6,0.4]).
:- set_sw(tr(rain),[0.2,0.8]).

weather(W,Time) :-
  Time >= 0,
  msw(init,W0),
  w(0,Time,W0,W).

w(T,T,W,W).
w(Now,T,WNow,WT) :-
    Now < T,
    msw(tr(WNow),WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

Weather in PRISM

```
values(init,[sun,rain]).
values(tr(_),[sun,rain])
```

random variables and their values

```
:- set_sw(init,[0.5,0.5]).
:- set_sw(tr(sun),[0.6,0.4]).
:- set_sw(tr(rain),[0.2,0.8]).

weather(W,Time) :-
  Time >= 0,
  msw(init,W0),
  w(0,Time,W0,W).

w(T,T,W,W).
w(Now,T,WNow,WT) :-
    Now < T,
    msw(tr(WNow),WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

Weather in PRISM

```
values(init,[sun,rain]).
values(tr(_),[sun,rain])
```

random variables and their values

```
:- set_sw(init,[0.5,0.5]).
:- set_sw(tr(sun),[0.6,0.4]).
:- set_sw(tr(rain),[0.2,0.8]).
```

probability distributions

```
weather(W,Time) :-
  Time >= 0,
  msw(init,W0),
  w(0,Time,W0,W).

w(T,T,W,W).
w(Now,T,WNow,WT) :-
    Now < T,
    msw(tr(WNow),WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

Weather in PRISM

```
values(init,[sun,rain]).
values(tr(_),[sun,rain])
```

random variables and their values

```
:- set_sw(init,[0.5,0.5]).
:- set_sw(tr(sun),[0.6,0.4]).
:- set_sw(tr(rain),[0.2,0.8]).
```

probability distributions

```
weather(W,Time) :-
  Time >= 0,
  msw(init,W0),
  w(0,Time,W0,W).

w(T,T,W,W).
w(Now,T,WNow,WT) :-
    Now < T,
    msw(tr(WNow),WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

set **`W0`** to random value of **`init`**

Weather in PRISM

```
values(init,[sun,rain]).
values(tr(_),[sun,rain])
```

random variables and their values

```
:- set_sw(init,[0.5,0.5]).
:- set_sw(tr(sun),[0.6,0.4]).
:- set_sw(tr(rain),[0.2,0.8]).
```

probability distributions

```
weather(W,Time) :-
  Time >= 0,
  msw(init,W0),
  w(0,Time,W0,W).

w(T,T,W,W).
w(Now,T,WNow,WT) :-
   Now < T,
   msw(tr(WNow),WNext),
   Next is Now+1,
   w(Next,T,WNext,WT).
```

set **`W0`** to random value of **`init`**

set **`WNext`** to random value of **`tr(WNow)`**, using **fresh** value on every call

Weather in PRISM / ProbLog

```
values(init,[sun,rain]).
values(tr(_),[sun,rain]).

:- set_sw(init,[0.5,0.5]).
:- set_sw(tr(sun),[0.6,0.4]).
:- set_sw(tr(rain),[0.2,0.8]).

weather(W,Time) :-
  Time >= 0,
  msw(init,W0),
  w(0,Time,W0,W).

w(T,T,W,W).
w(Now,T,WNow,WT) :-
    Now < T,
    msw(tr(WNow),WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

```
0.5::init(sun) ;
  0.5::init(rain) <- true.
0.6::tr(T,sun,sun) ;
  0.4::tr(T,sun,rain) <- true.
0.2::tr(T,rain,sun) ;
  0.8::tr(T,rain,rain) <- true.

weather(W,Time) :-
  Time >= 0,
  init(W0),
  w(0,Time,W0,W).

w(T,T,W,W).
w(Now,T,WNow,WT) :-
    Now < T,
    tr(Now,WNow,WNext),
    Next is Now+1,
    w(Next,T,WNext,WT).
```

ProbLog needs to explicitly use different facts at each call

Probabilistic Prologs: Two Views

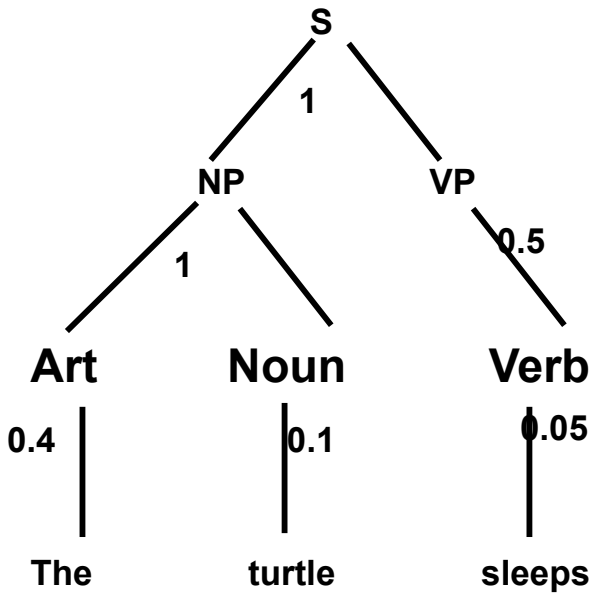
- Distribution semantics:
  - probability distribution over interpretations
  - degree of belief
- Stochastic Logic Programs (SLPs):
  - probability distribution over query answers
  - like in probabilistic grammars

Probabilistic Prologs: Two Views

- Distribution semantics:
  - probability distribution over interpretations
  - degree of belief
- Stochastic Logic Programs (SLPs):
  - probability distribution over query answers
  - like in probabilistic grammars

Probabilistic Context Free Grammars

1.0 : S -> NP, VP
1.0 : NP -> Art, Noun
0.6 : Art -> a
0.4 : Art -> the
0.1 : Noun -> turtle
0.1 : Noun -> turtles
…
0.5 : VP -> Verb
0.5 : VP -> Verb, NP
0.05 : Verb -> sleep
0.05 : Verb -> sleeps
….



P(parse tree) = 1x1x.5x.1x.4x.05

PCFGs

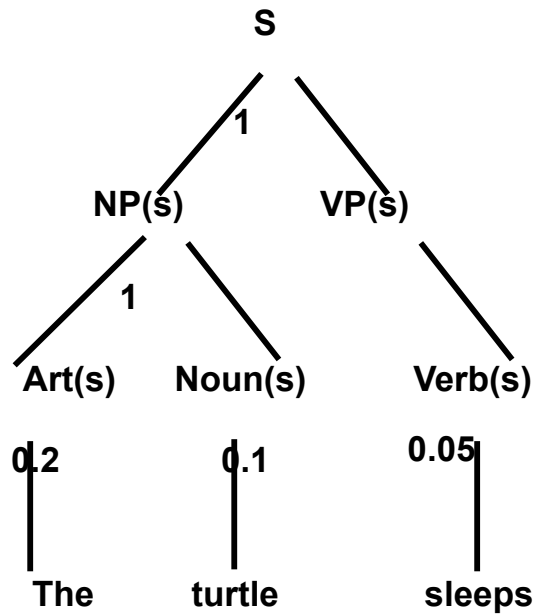
$P(parse\ tree) = \prod_i p_i^{c_i}$
where p_i is the probability of rule i
and c_i the number of times
it is used in the parse tree

$P(sentence) = \sum_{p:parsetree} P(p)$

Observe that derivations always succeed, that is
$S \to T, Q$ and $T \to R, U$
always yields
$S \to R, U, Q$

Probabilistic Definite Clause Grammar

1.0 S -> NP(Num), VP(Num)
1.0 NP(Num) -> Art(Num), Noun(Num)
0.6 Art(sing) -> a
0.2 Art(sing) -> the
0.2 Art(plur) -> the
0.1 Noun(sing) -> turtle
0.1 Noun(plur) -> turtles
…
0.5 VP(Num) -> Verb(Num)
0.5 VP(Num) -> Verb(Num), NP(Num)
0.05 Verb(sing) -> sleeps
0.05 Verb(plur) -> sleep
….



P(derivation tree) = 1x1x.5x.1x .2 x.05

Stochastic Logic Programs

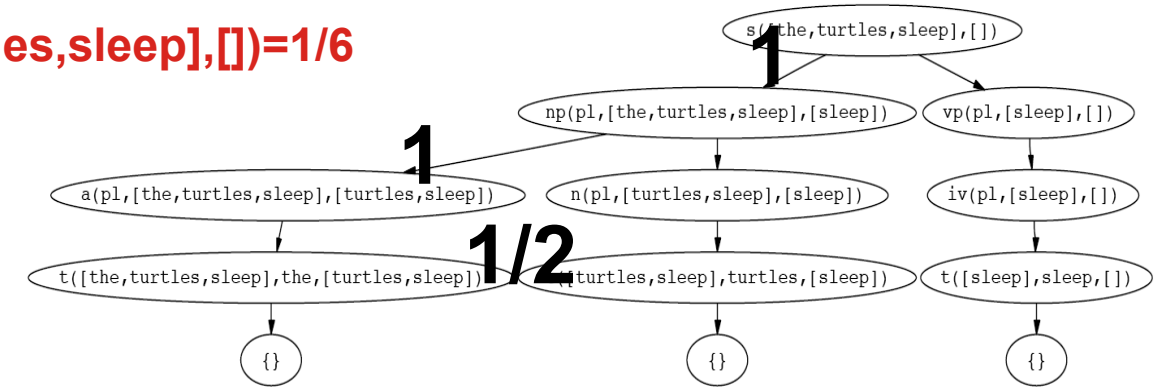
[Muggleton, Cussens]

In SLP notation

1
1/3
1/2

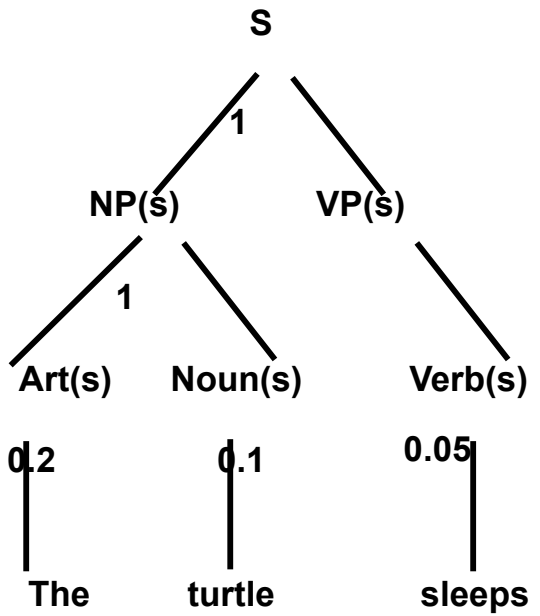
```
sentence(A, B) :- noun_phrase(C, A, D), verb_phrase(C, D, B).
noun_phrase(A, B, C) :- article(A, B, D), noun(A, D, C).
verb_phrase(A, B, C) :- intransitive_verb(A, B, C).
article(singular, A, B) :- terminal(A, a, B).
article(singular, A, B) :- terminal(A, the, B).
article(plural, A, B)   :- terminal(A, the, B).
noun(singular, A, B) :- terminal(A, turtle, B).
noun(plural, A, B)   :- terminal(A, turtles, B).
intransitive_verb(singular, A, B) :- terminal(A, sleeps, B).
intransitive_verb(plural, A, B) :- terminal(A, sleep, B).
terminal([A|B],A,B).
```

P(s([the,turtles,sleep],[])=1/6



Probabilistic DCG

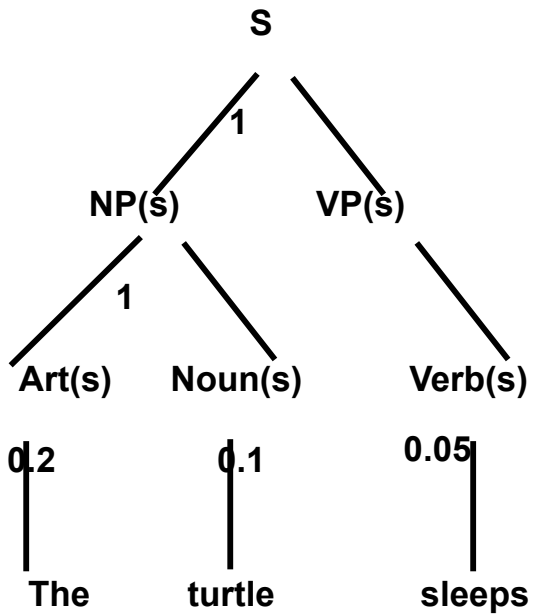
1.0 S -> NP(Num), VP(Num)
1.0 NP(Num) -> Art(Num), Noun(Num)
0.6 Art(sing) -> a
0.2 Art(sing) -> the
0.2 Art(plur) -> the
0.1 Noun(sing) -> turtle
0.1 Noun(plur) -> turtles
…
0.5 VP(Num) -> Verb(Num)
0.5 VP(Num) -> Verb(Num), NP(Num)
0.05 Verb(sing) -> sleeps
0.05 Verb(plur) -> sleep
….



P(derivation tree) = 1x1x.5x.1x .2 x.05

Probabilistic DCG

1.0 S -> NP(Num), VP(Num)
1.0 NP(Num) -> Art(Num), Noun(Num)
0.6 Art(sing) -> a
0.2 Art(sing) -> the
0.2 Art(plur) -> the
0.1 Noun(sing) -> turtle
0.1 Noun(plur) -> turtles
…
0.5 VP(Num) -> Verb(Num)
0.5 VP(Num) -> Verb(Num), NP(Num)
0.05 Verb(sing) -> sleeps
0.05 Verb(plur) -> sleep
….



P(derivation tree) = 1x1x.5x.1x .2 x.05

What about "A turtles sleeps" ?

SLPs

$P_d(derivation) = \prod_i p_i^{c_i}$
where p_i is the probability of rule i
and c_i the number of times
it is used in the parse tree

Observe that some derivations now fail due to unification,
$np(Num) \rightarrow art(Num), noun(Num)$ and $art(sing) \rightarrow a$
$noun(plural) \rightarrow turtles$

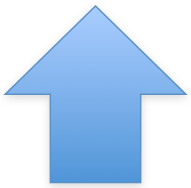
Normalization necessary
$P_s(proof) = \frac{P_d(proof)}{\sum_i P_d(proof_i)}$

ProPPR

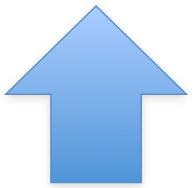
- A variation on SLPs
- Integrating concepts from Personalized Page Rank
- Fast inference and rule learning abilities
- Used by CMU group for NELL (Never Ending Learning)
- See [Wang et al., CIKM 13, arXiv:1404.3301]

Sample ProPPR program….

```
about(X,Z) :- handLabeled(X,Z)          # base.
about(X,Z) :- sim(X,Y),about(Y,Z)       # prop.
sim(X,Y) :- links(X,Y)                  # sim,link.
sim(X,Y) :-
    hasWord(X,W),hasWord(Y,W),
    linkedBy(X,Y,W)                     # sim,word.
linkedBy(X,Y,W) :- true                 # by(W).
```

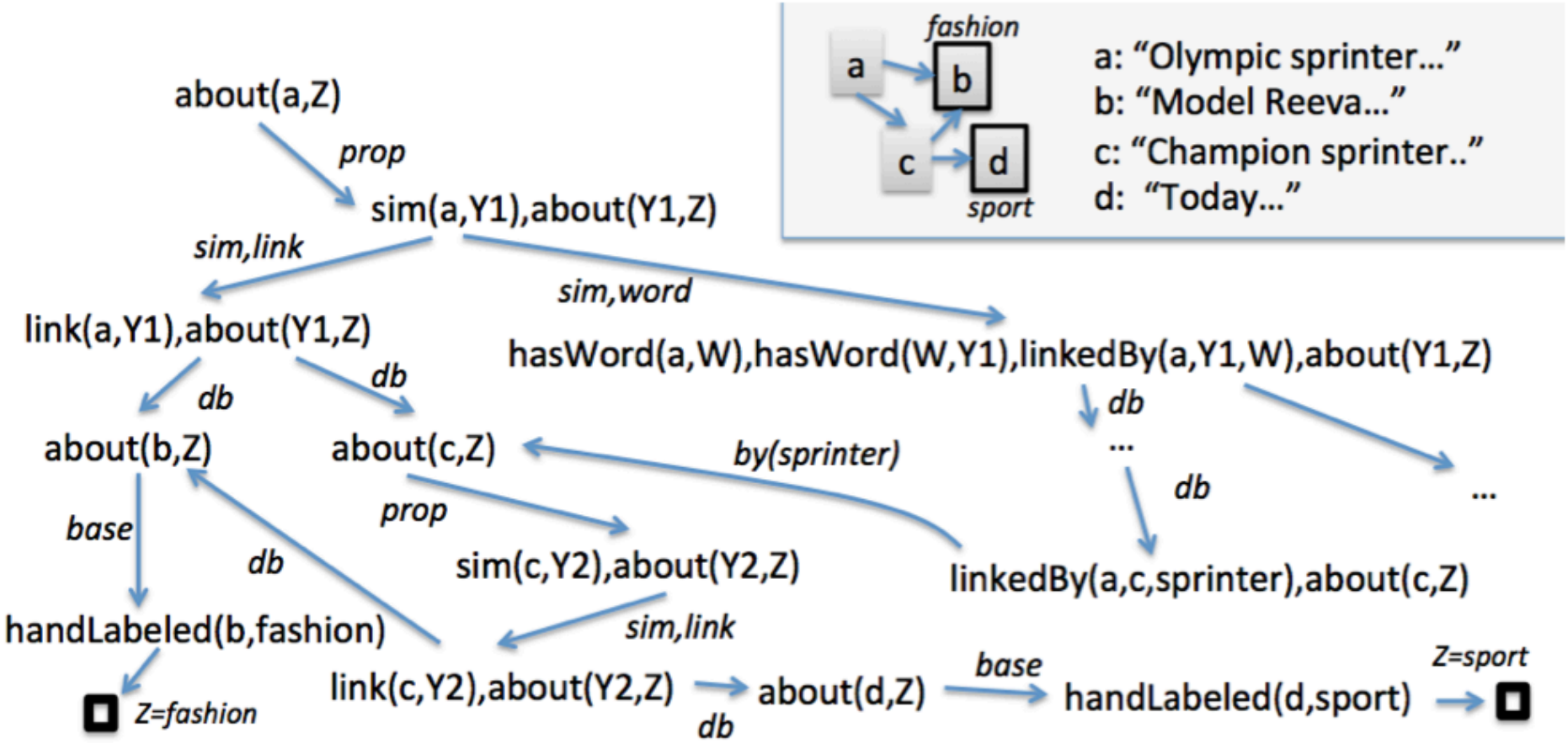


Horn rules



features of rules
(vars from head ok)

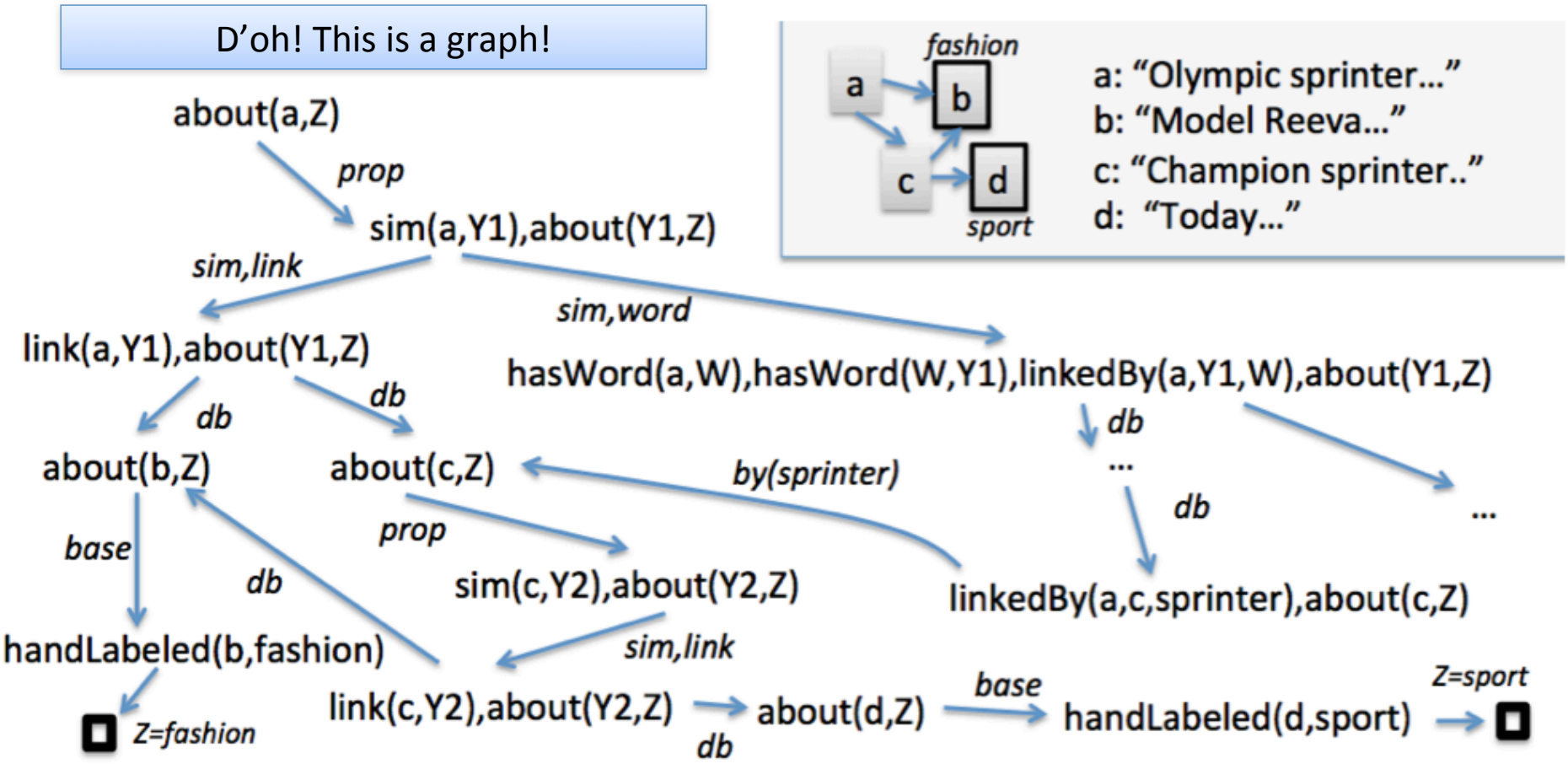
[Slide by William Cohen]



.. and search space…

[Slide by William Cohen]

```
about(X,Z) :- handLabeled(X,Z)          # base.
about(X,Z) :- sim(X,Y),about(Y,Z)       # prop.
sim(X,Y) :- links(X,Y)                  # sim,link.
sim(X,Y) :-
    hasWord(X,W),hasWord(Y,W),
    linkedBy(X,Y,W)                     # sim,word.
linkedBy(X,Y,W) :- true                 # by(W).
```

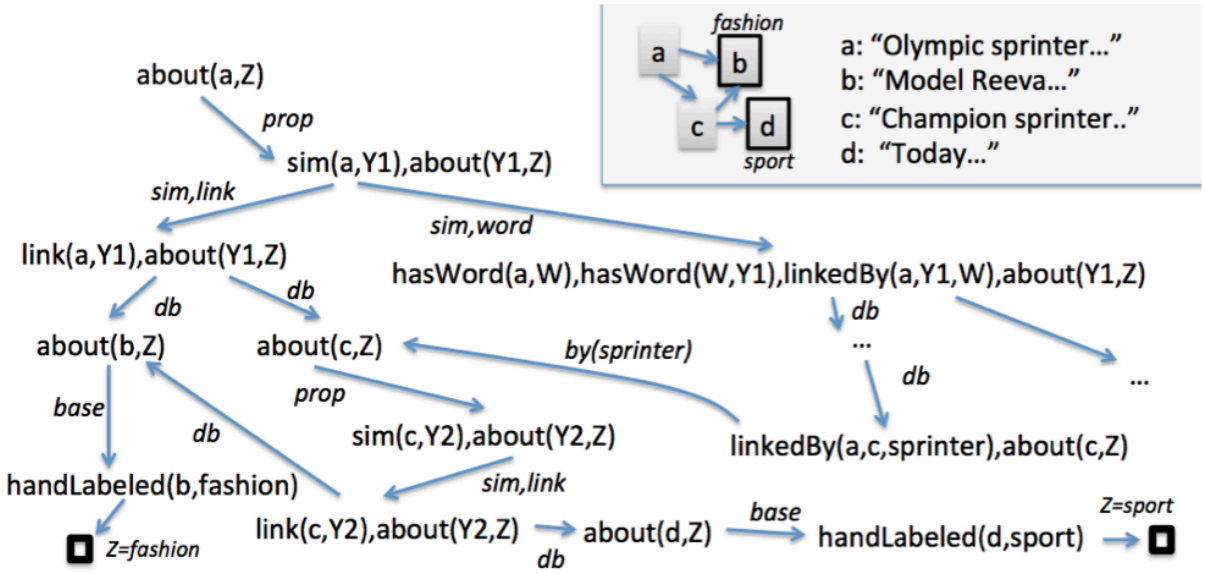


.. and search space…

[Slide by William Cohen]

```
about(X,Z) :- handLabeled(X,Z)          # base.
about(X,Z) :- sim(X,Y),about(Y,Z)       # prop.
sim(X,Y) :- links(X,Y)                  # sim,link.
sim(X,Y) :-
    hasWord(X,W),hasWord(Y,W),
    linkedBy(X,Y,W)                     # sim,word.
linkedBy(X,Y,W) :- true                 # by(W).
```

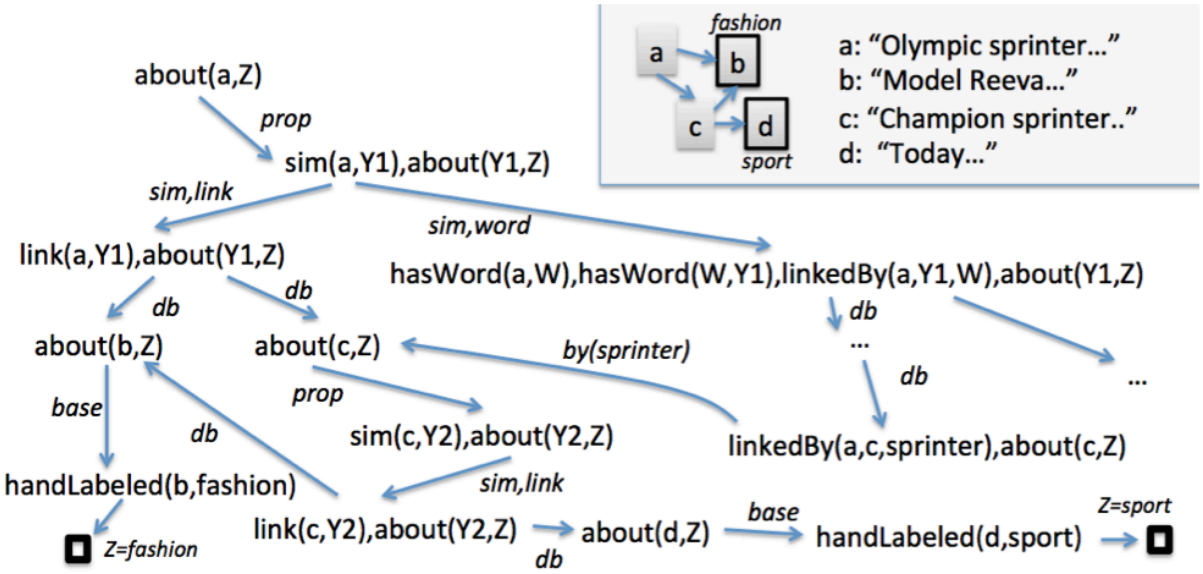
- Score for a query soln (e.g., "Z=sport" for "about(a,Z)") depends on *probability* of reaching a □ node*
  - *learn* transition probabilities based on features of the rules
  - implicit "reset" transitions with (p≥α) back to query node
- Looking for answers supported by *many short proofs*



[Slide by William Cohen]

- Score for a query soln (e.g., "Z=sport" for "about(a,Z)") depends on *probability* of reaching a □ node*
  - *learn* transition probabilities based on features of the rules
  - implicit "reset" transitions with (p≥α) back to query node
- Looking for answers supported by *many short proofs*

*Exactly as in Stochastic Logic Programs [Cussens, 2001]

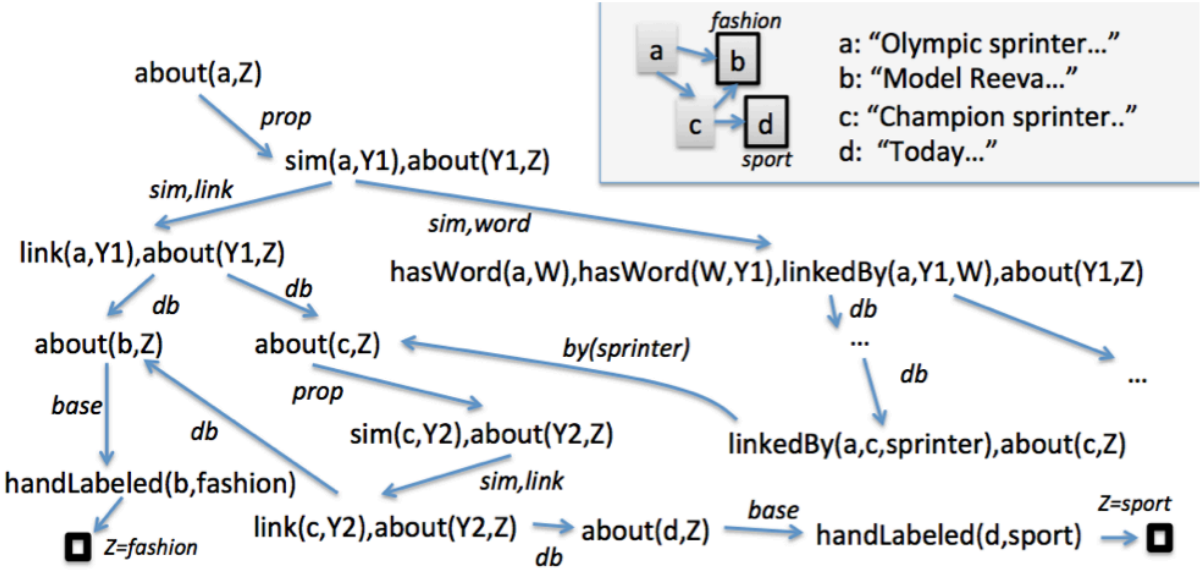


[Slide by William Cohen]

- Score for a query soln (e.g., "Z=sport" for "about(a,Z)") depends on *probability* of reaching a □ node*
  - *learn* transition probabilities based on features of the rules
  - implicit "reset" transitions with (p≥α) back to query node
- Looking for answers supported by *many short proofs*

"Grounding" size is O(1/αε) … ie *independent* of DB size ➔ fast approx incremental inference (Reid,Lang,Chung, 08)

*Exactly as in Stochastic Logic Programs [Cussens, 2001]



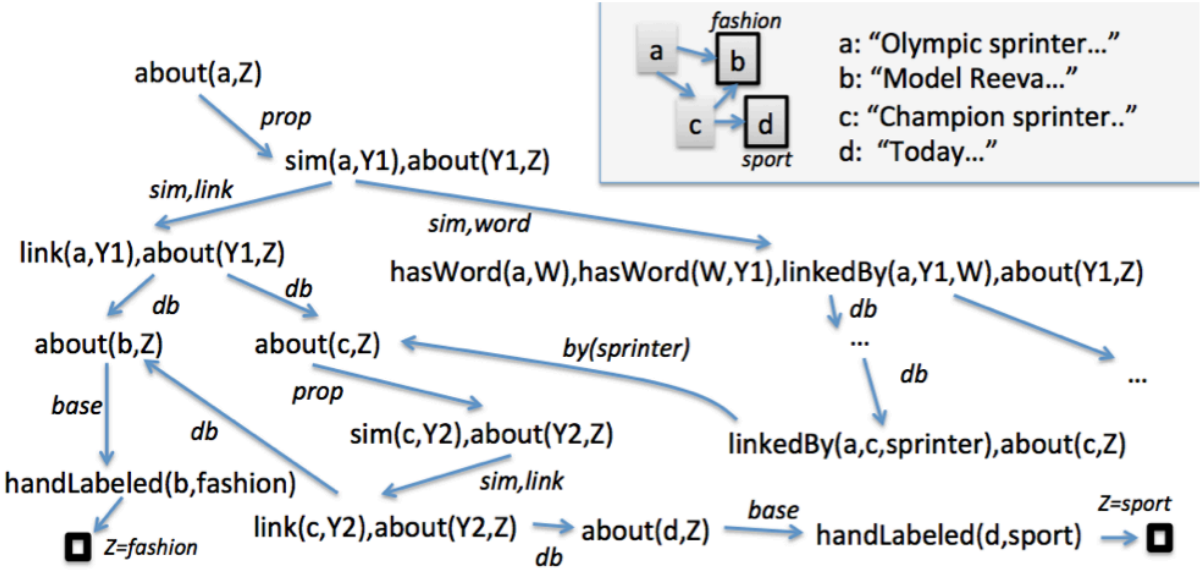
[Slide by William Cohen]

- Score for a query soln (e.g., "Z=sport" for "about(a,Z)") depends on *probability* of reaching a □ node*
  - *learn* transition probabilities based on features of the rules
  - implicit "reset" transitions with (p≥α) back to query node
- Looking for answers supported by *many short proofs*

"Grounding" size is O(1/αε) … ie *independent* of DB size ➔ fast approx incremental inference (Reid,Lang,Chung, 08)

*Exactly as in Stochastic Logic Programs [Cussens, 2001]

Learning: supervised variant of personalized PageRank (Backstrom & Leskovic, 2011)



[Slide by William Cohen]

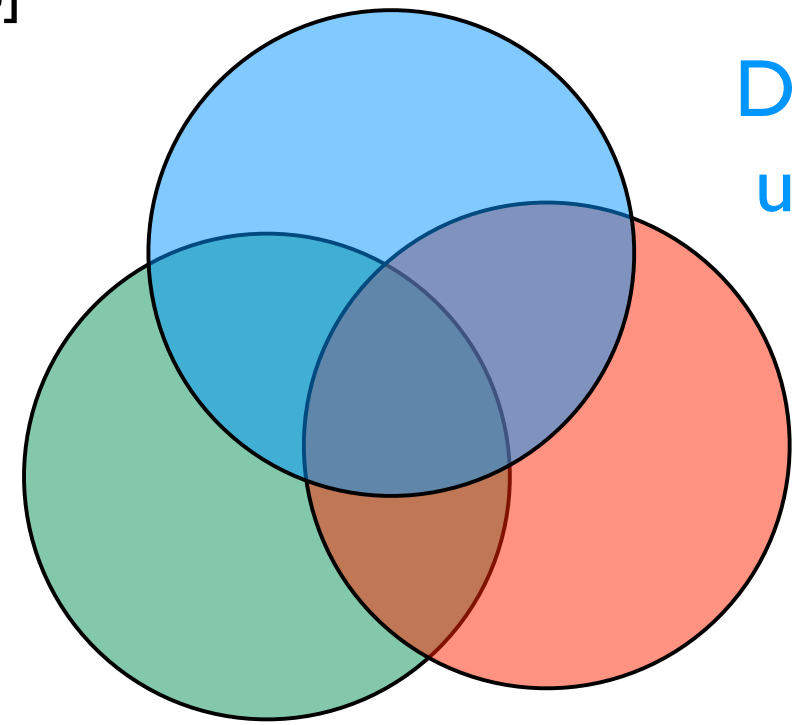
Probabilistic Programming Languages outside LP

- IBAL [Pfeffer 01]
- Figaro [Pfeffer 09]
- Church [Goodman et al 08]
- BLOG [Milch et al 05]
- and many more appearing recently

Church

probabilistic functional programming

[Goodman et al, UAI 08]



Dealing with uncertainty

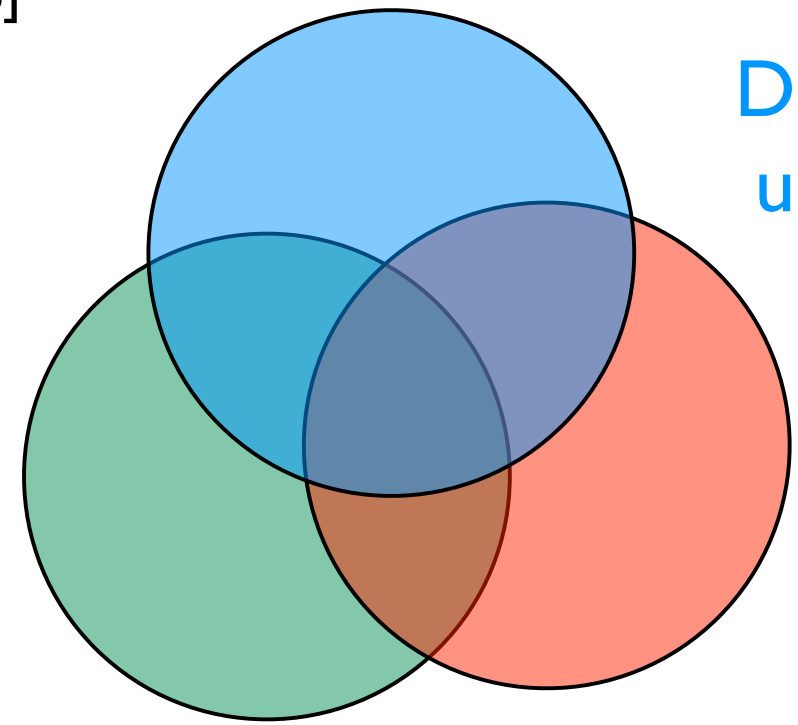
Reasoning with relational data

Learning

Church

probabilistic functional programming

[Goodman et al, UAI 08]

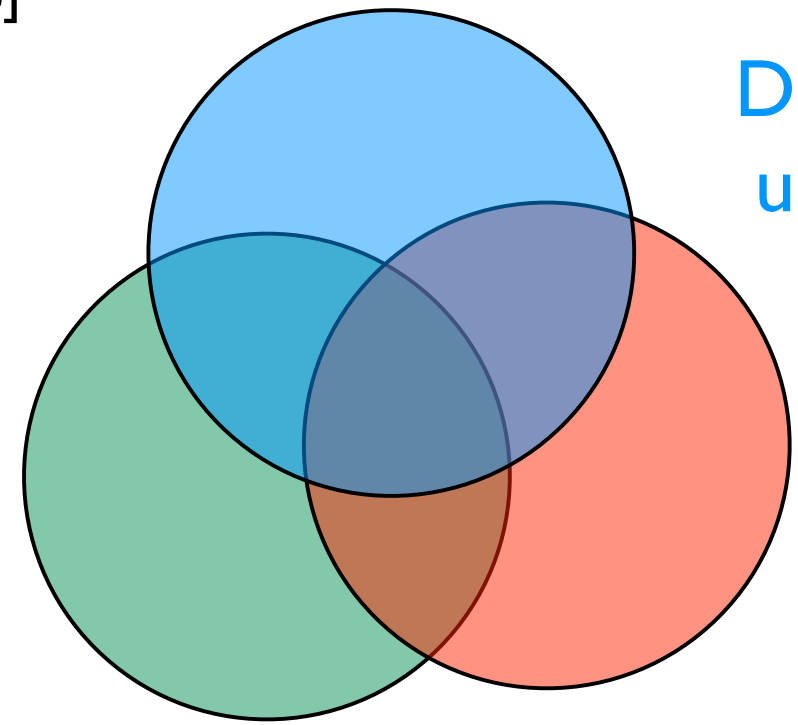



```
(define plus5 (lambda (x) (+ x 5)))

(map plus5 '(1 2 3))
```

Church

probabilistic functional programming

[Goodman et al, UAI 08]



Dealing with uncertainty

functional programming

Learning

one execution

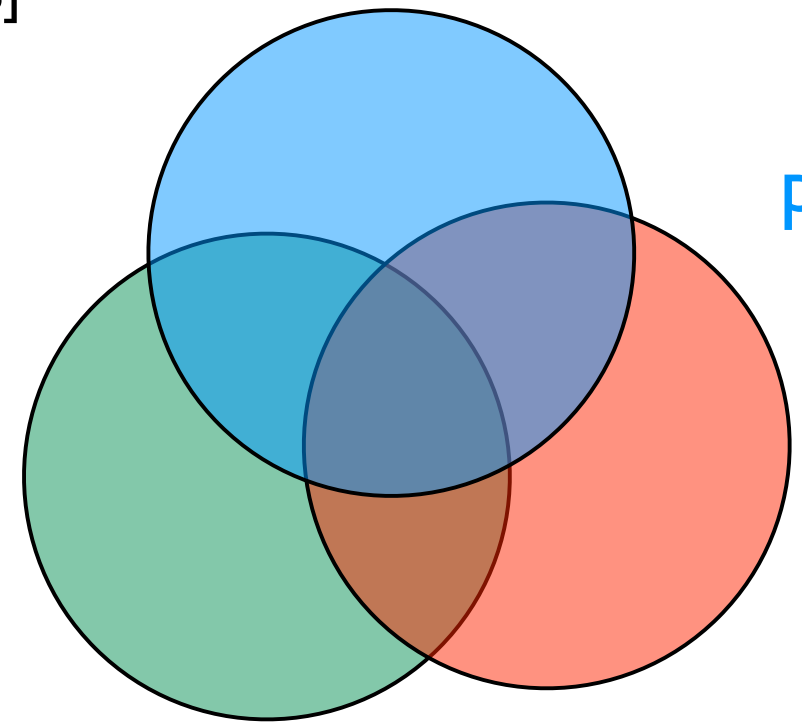
```
(define plus5 (lambda (x) (+ x 5)))

(map plus5 '(1 2 3))
```

Church

probabilistic functional programming

[Goodman et al, UAI 08]

```
(define randplus5
 (lambda (x) (if (flip 0.6)
                 (+ x 5)
                 x)))

(map randplus5 '(1 2 3))
```



random primitives

functional programming

Learning

one execution

```
(define plus5 (lambda (x) (+ x 5)))

(map plus5 '(1 2 3))
```

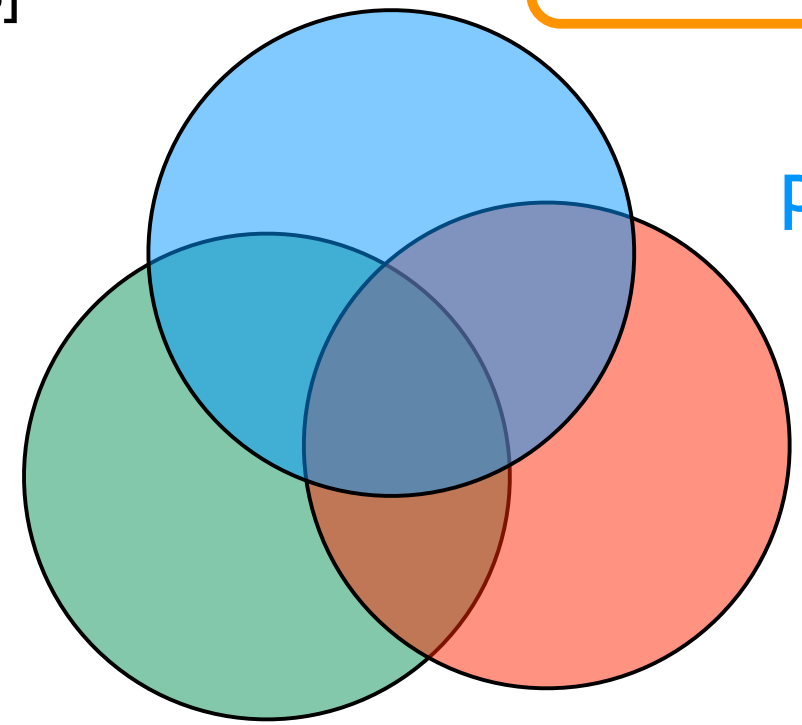
Church

probabilistic functional programming

[Goodman et al, UAI 08]

several possible executions

```
(define randplus5
 (lambda (x) (if (flip 0.6)
                 (+ x 5)
                 x)))

(map randplus5 '(1 2 3))
```



random primitives

functional programming

Learning

one execution

```
(define plus5 (lambda (x) (+ x 5)))

(map plus5 '(1 2 3))
```

Church

probabilistic functional programming

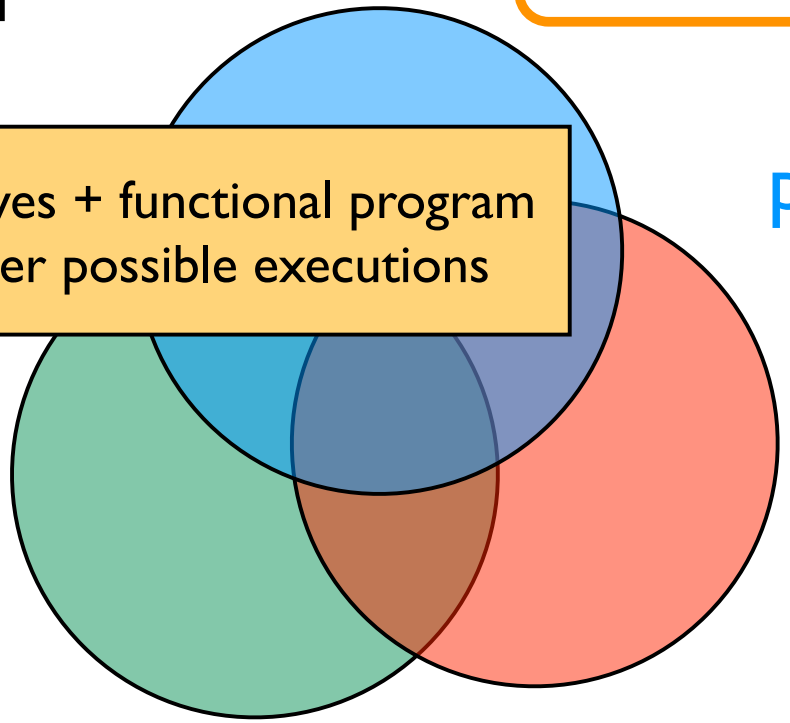
[Goodman et al, UAI 08]

several possible executions

```
(define randplus5
 (lambda (x) (if (flip 0.6)
                 (+ x 5)
                 x)))

(map randplus5 '(1 2 3))
```

random primitives

probabilistic primitives + functional program
→ distribution over possible executions



functional programming

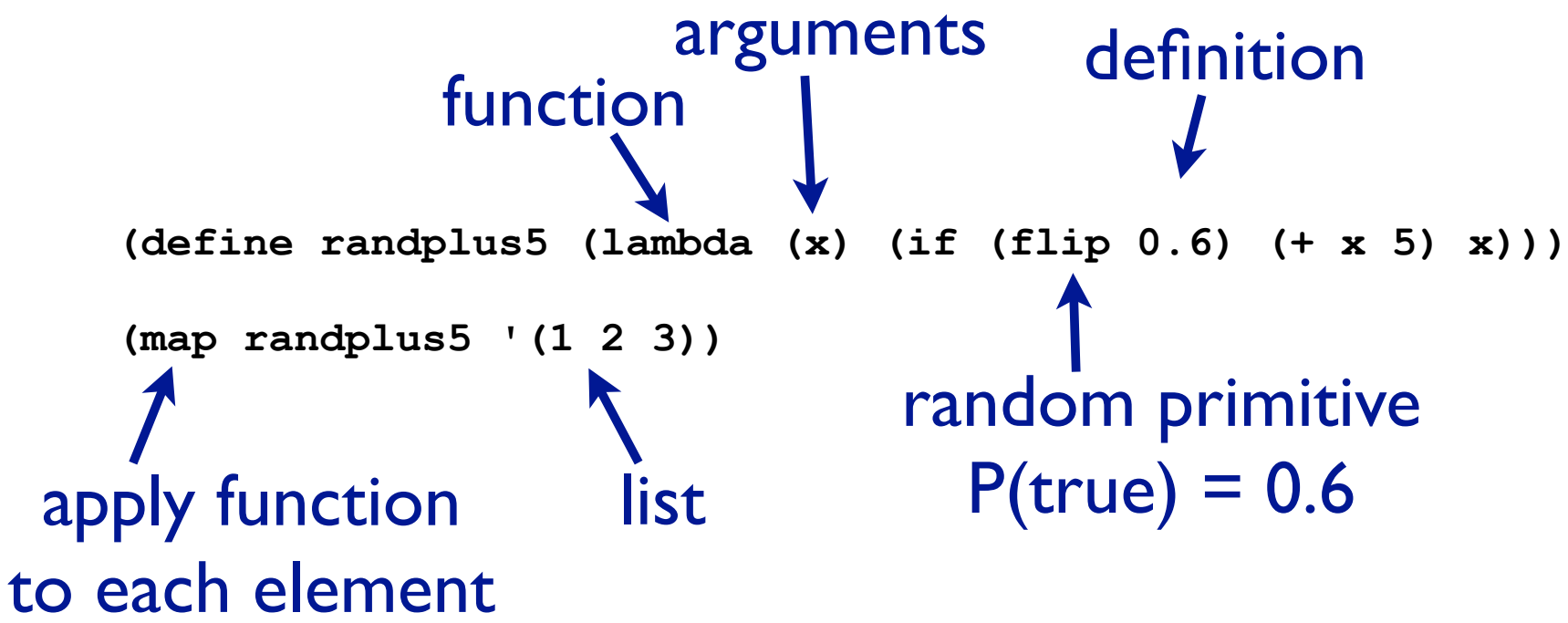
Learning

one execution

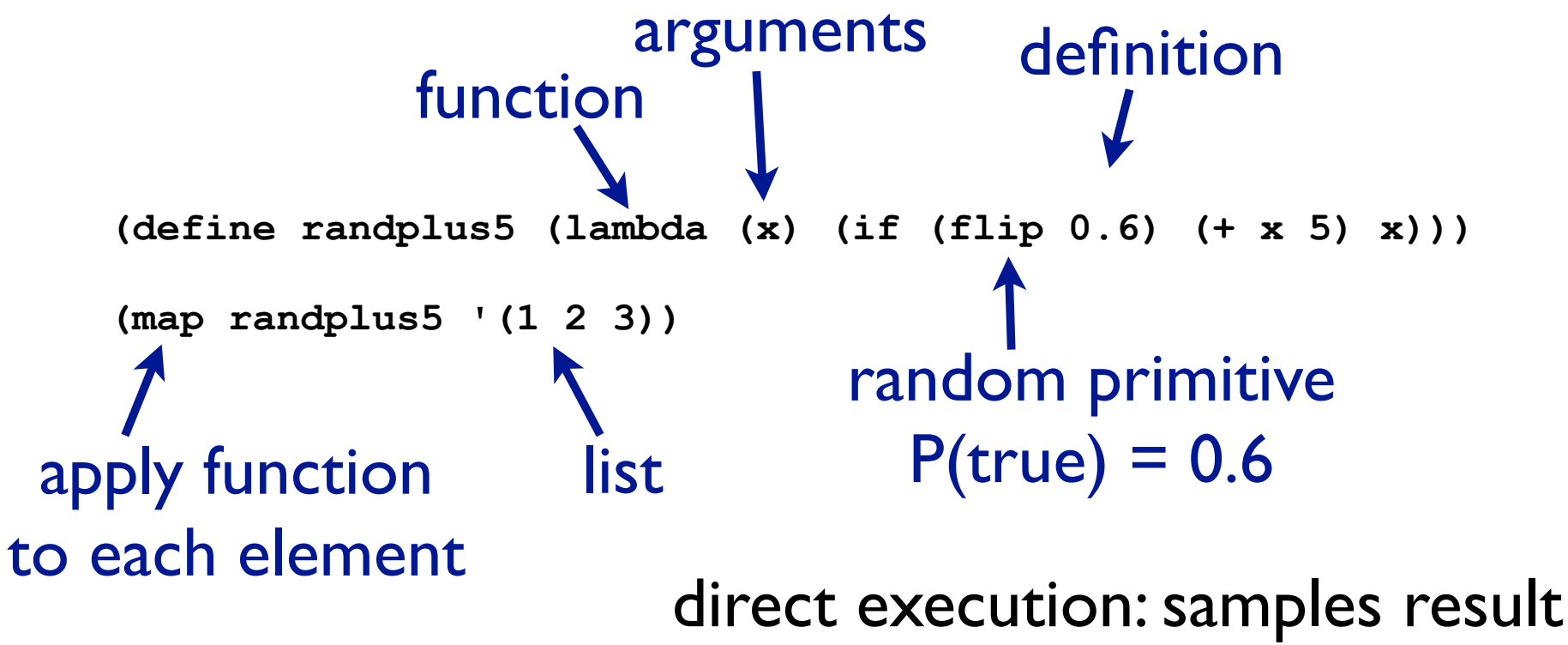
```
(define plus5 (lambda (x) (+ x 5)))

(map plus5 '(1 2 3))
```


http://probmods.org

Church Example

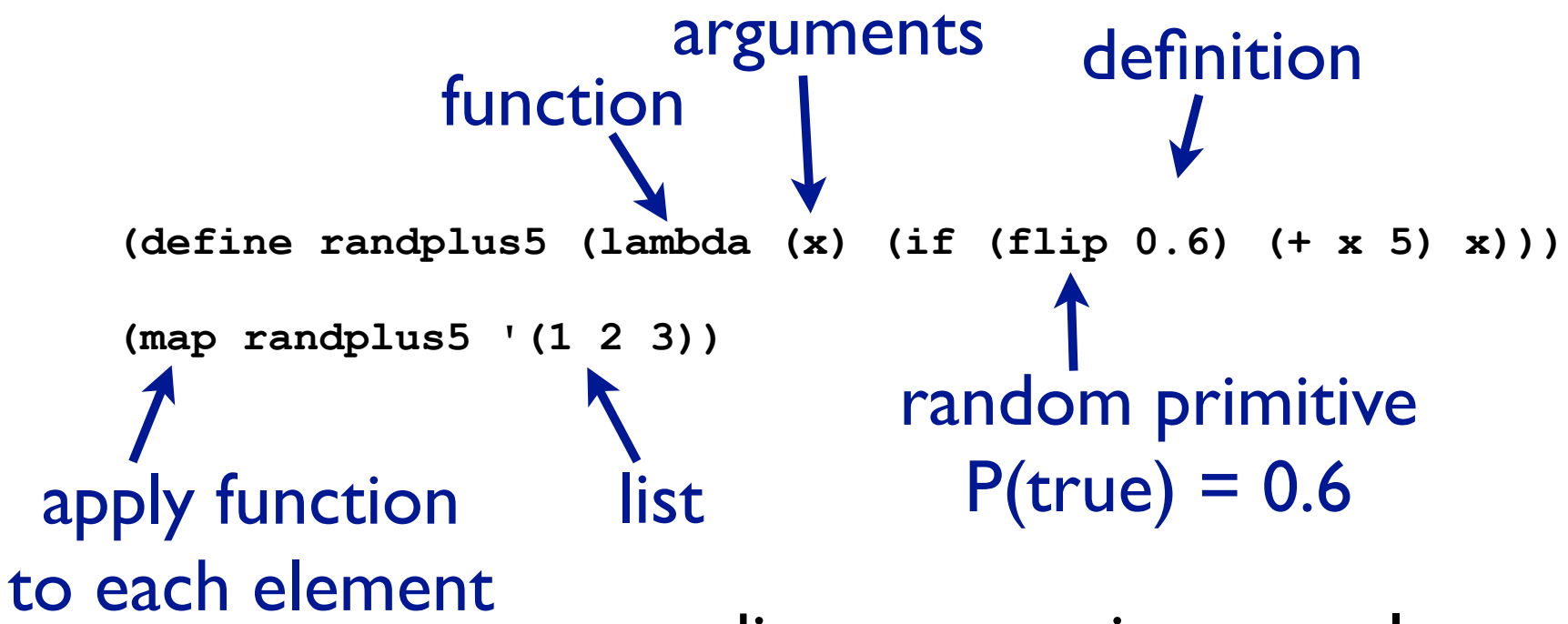


Church Example



direct execution: samples result

Church Example



direct execution: samples result

sampling also supports continuous RVs, e.g.,
`(* (gaussian 0 1) (gaussian 0 1))`

Computing probability distribution

```
(
enumeration-query

(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 2))

true
)
```

enumerates all executions & sums probabilities per result

query

evidence

(((1 2) (1 7) (6 2) (6 7)) (0.16 0.24 0.24 0.36))

in ProbLog?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 2))
```

in ProbLog?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 2))
```

```
0.4::p5(N,N);0.6::p5(N,M) <- M is N+5.
lp5([],[]).
lp5([N|L],[M|K]) :-
    p5(N,M),
    lp5(L,K).

query(lp5([1,2],_)).
```

in ProbLog?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 2))
```

```
0.4::p5(N,N);0.6::p5(N,M) <- M is N+5.
lp5([],[]).
lp5([N|L],[M|K]) :-
    p5(N,M),
    lp5(L,K).

query(lp5([1,2],_)).
```

what if the list is [1,1]?

in ProbLog?

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 2))
```

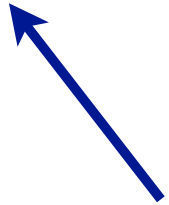
```
0.4::p5(N,N);0.6::p5(N,M) <- M is N+5.
lp5([],[]).
lp5([N|L],[M|K]) :-
    p5(N,M),
    lp5(L,K).

query(lp5([1,2],_)).
```

what if the list is [1,1]?

```
0.4::p5(N,N,ID);0.6::p5(N,M,ID) <- M is N+5.
lp5([],[]).
lp5([N|L],[M|K]) :-
    p5(N,M,L),
    lp5(L,K).

query(lp5([1,2],_)).
```

Stochastic Memoization

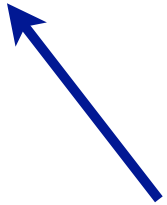
```
(define randplus5 (mem (lambda (x) (if (flip 0.6) (+ x 5) x))))

(map randplus5 '(1 1))
```



remember first value &
reuse for all later calls

Stochastic Memoization

```
(define randplus5 (mem (lambda (x) (if (flip 0.6) (+ x 5) x))))

(map randplus5 '(1 1))
```

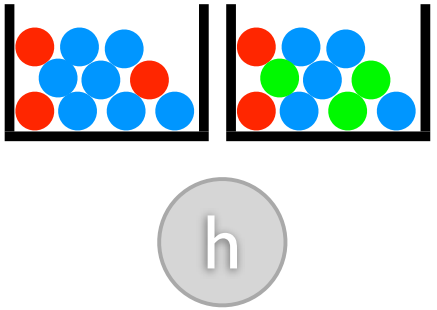


remember first value &
reuse for all later calls

ProbLog always memoizes
PRISM never memoizes
Church allows fine-grained choice

Church by example:

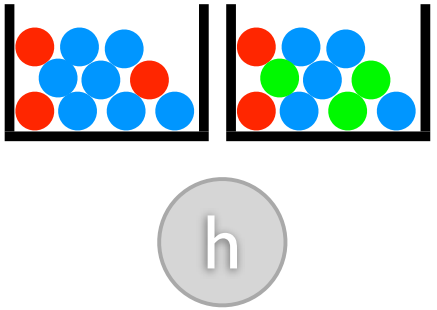
A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

Church by example:

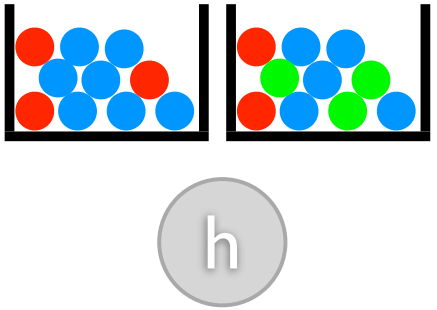
A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

Church by example:

A bit of gambling

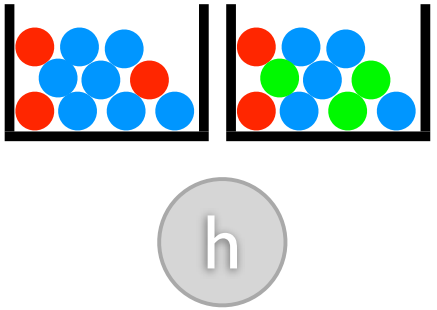


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))
```

Church by example:

A bit of gambling

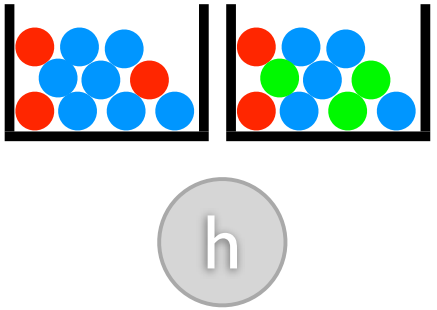


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))
```

Church by example:

A bit of gambling

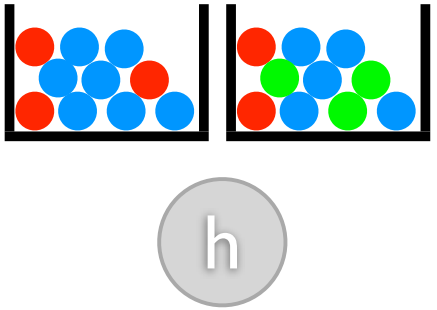


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))
```

Church by example:

A bit of gambling

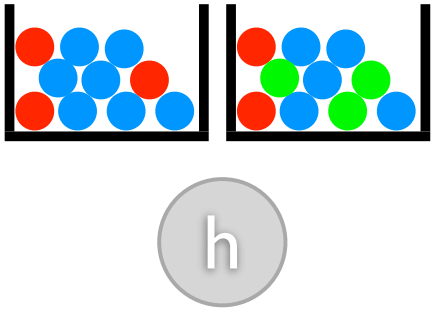


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                 (multinomial '(red green blue) '(0.2 0.3 0.5)))))
```

Church by example:

A bit of gambling

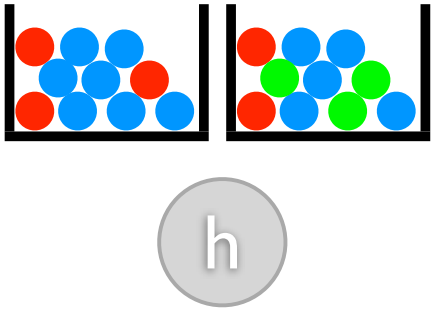


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                  (multinomial '(red green blue) '(0.2 0.3 0.5)))))
```

Church by example:

A bit of gambling

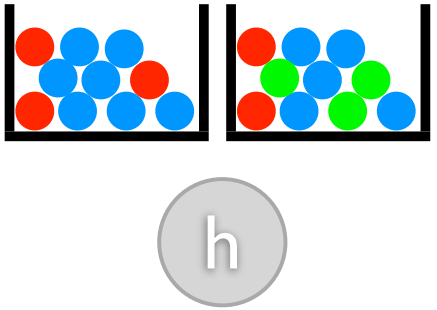


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                  (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))
```

Church by example:

A bit of gambling

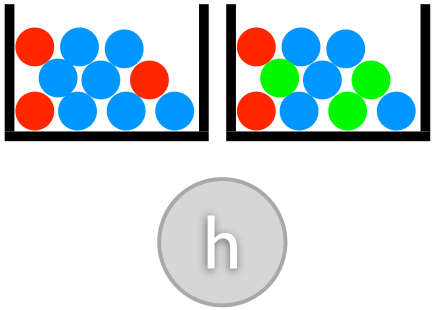


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                  (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))
```

Church by example:

A bit of gambling

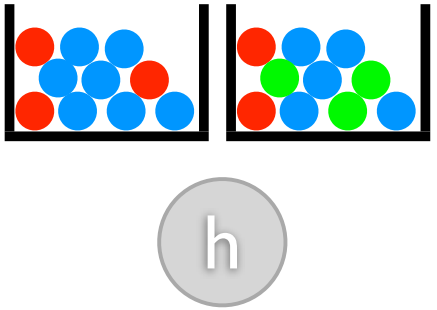


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                  (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))
```

Church by example:

A bit of gambling

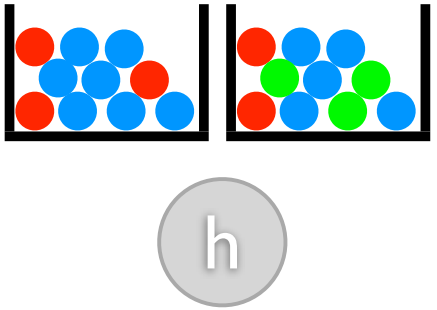


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                  (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))

(define win2 (equal? (color1) (color2)))
```

Church by example:

A bit of gambling

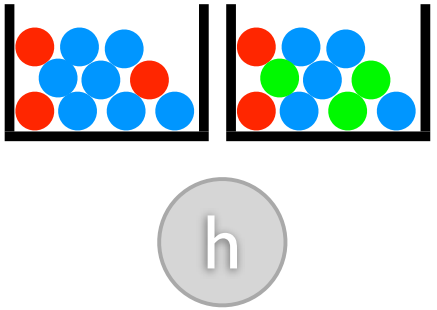


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                  (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))

(define win2 (equal? (color1) (color2)))
```

Church by example:

A bit of gambling

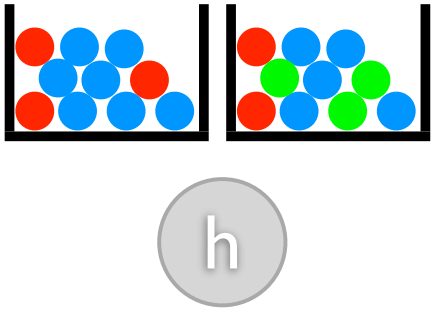


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                  (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))

(define win2 (equal? (color1) (color2)))

(define win (or win1 win2))
```

Church by example:

A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                  (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))

(define win2 (equal? (color1) (color2)))

(define win (or win1 win2))
```

Sampling execution

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                 (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))

(define win2 (equal? (color1) (color2)))

(define win (or win1 win2))
```

```
win
```

query

Marginals via enumeration

```
(enumeration-query
  (define heads (mem (lambda () (flip 0.4))))
  (define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))
  (define color2 (mem (lambda ()
                        (multinomial '(red green blue) '(0.2 0.3 0.5)))))
  (define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))
  (define win1 (and (heads) redball))
  (define win2 (equal? (color1) (color2)))
  (define win (or win1 win2))

  win
  true )
```

query

evidence

Histogram via sampling

```
(repeat 1000 (lambda ()
           (rejection-query

  (define heads (mem (lambda () (flip 0.4))))

  (define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

  (define color2 (mem (lambda ()
                   (multinomial '(red green blue) '(0.2 0.3 0.5)))))

  (define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

  (define win1 (and (heads) redball))

  (define win2 (equal? (color1) (color2)))

  (define win (or win1 win2))

  win

  true ))))
```

query

evidence

Probabilistic Programming Summary

- Church: functional programming + random primitives
- probabilistic generative model
- stochastic memoization
- sampling
- increasing number of probabilistic programming languages using various underlying paradigms

ProbLog	**PRISM**	**Church**
probabilistic facts & choices	probabilistic choices	random primitives
all RVs memoized	no RVs memoized	user-defined per RV
Prolog	Prolog with mutually exclusive derivations	λ-calculus functions
distribution over worlds	distribution over derivations / answers	distribution over computations / answers

Roadmap

- Modeling
- Inference
- Learning & Dynamics
- Advanced Inference and KBMC

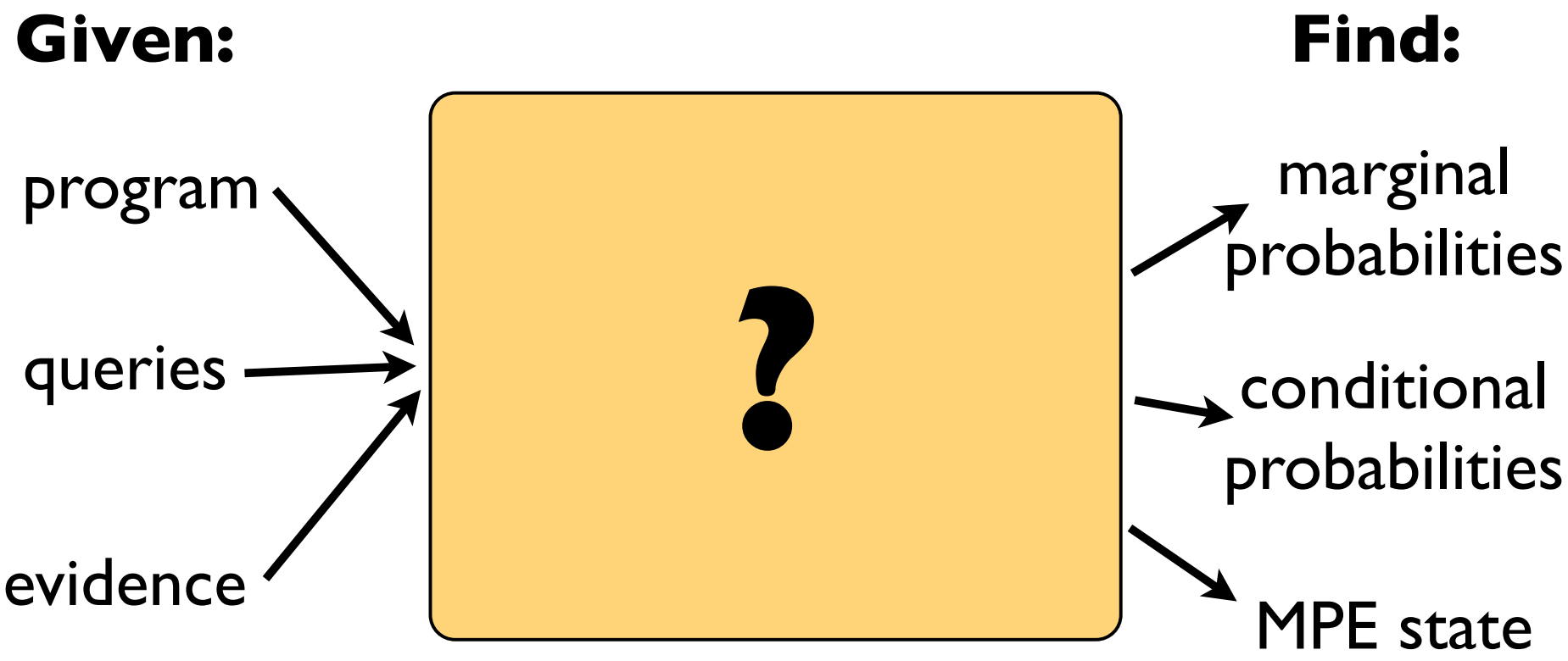
... with some detours on the way

PART II : Inference

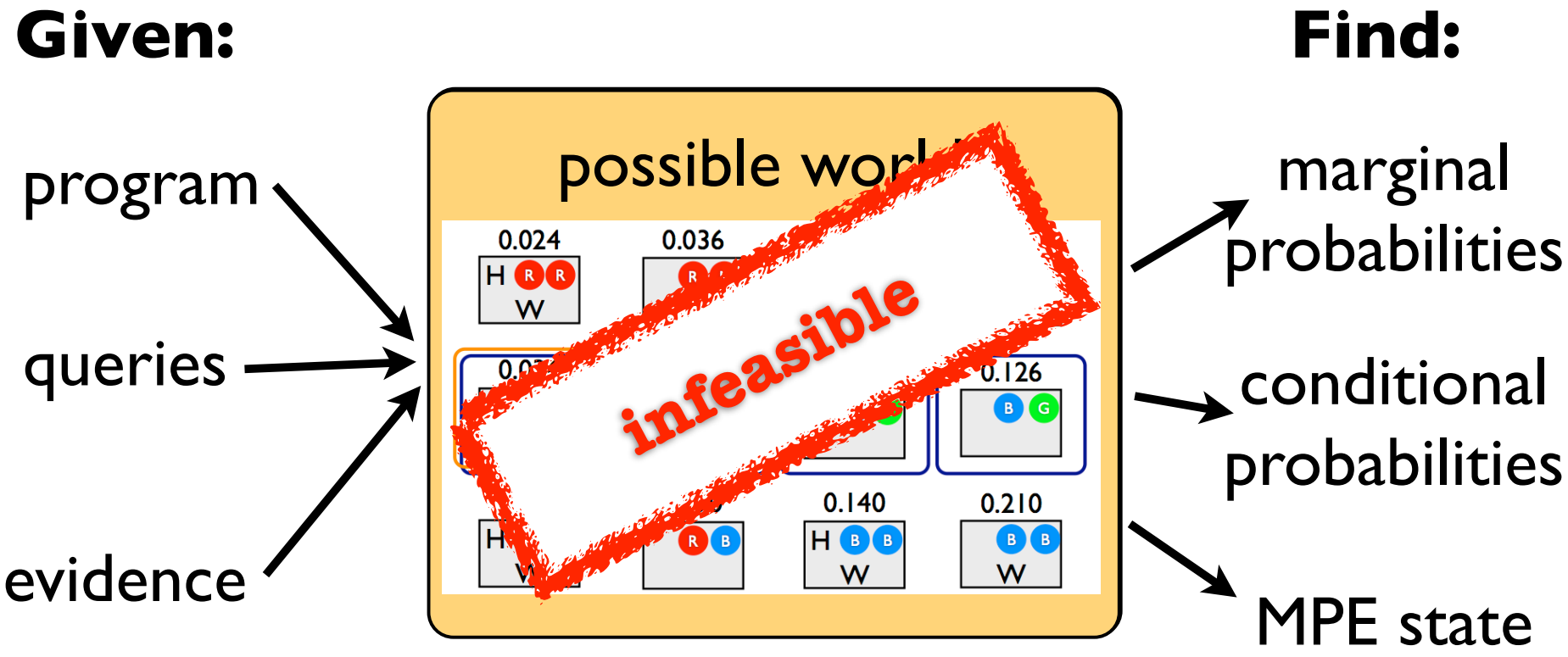
Reasoning

- Exact inference with knowledge compilation
  - using proofs
  - using models
  - in PRISM
- Approximate inference

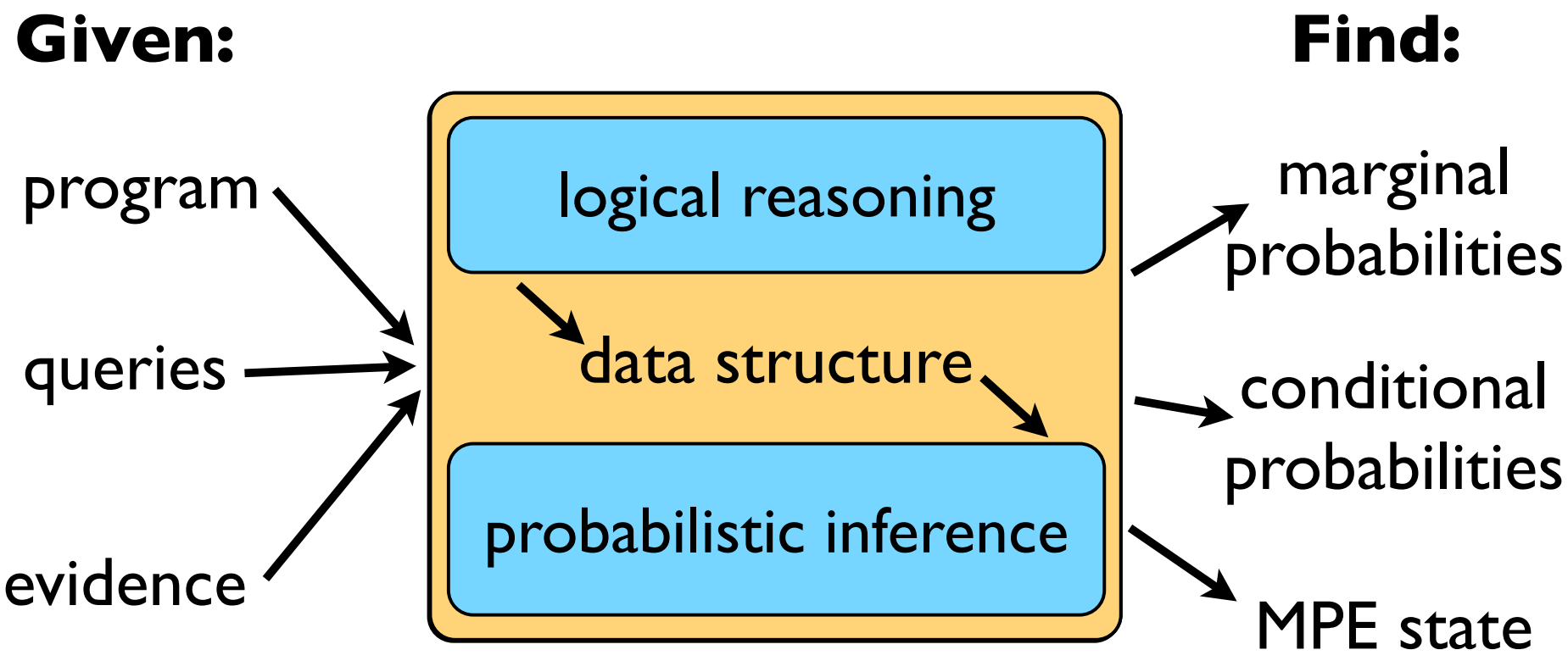
Answering Questions



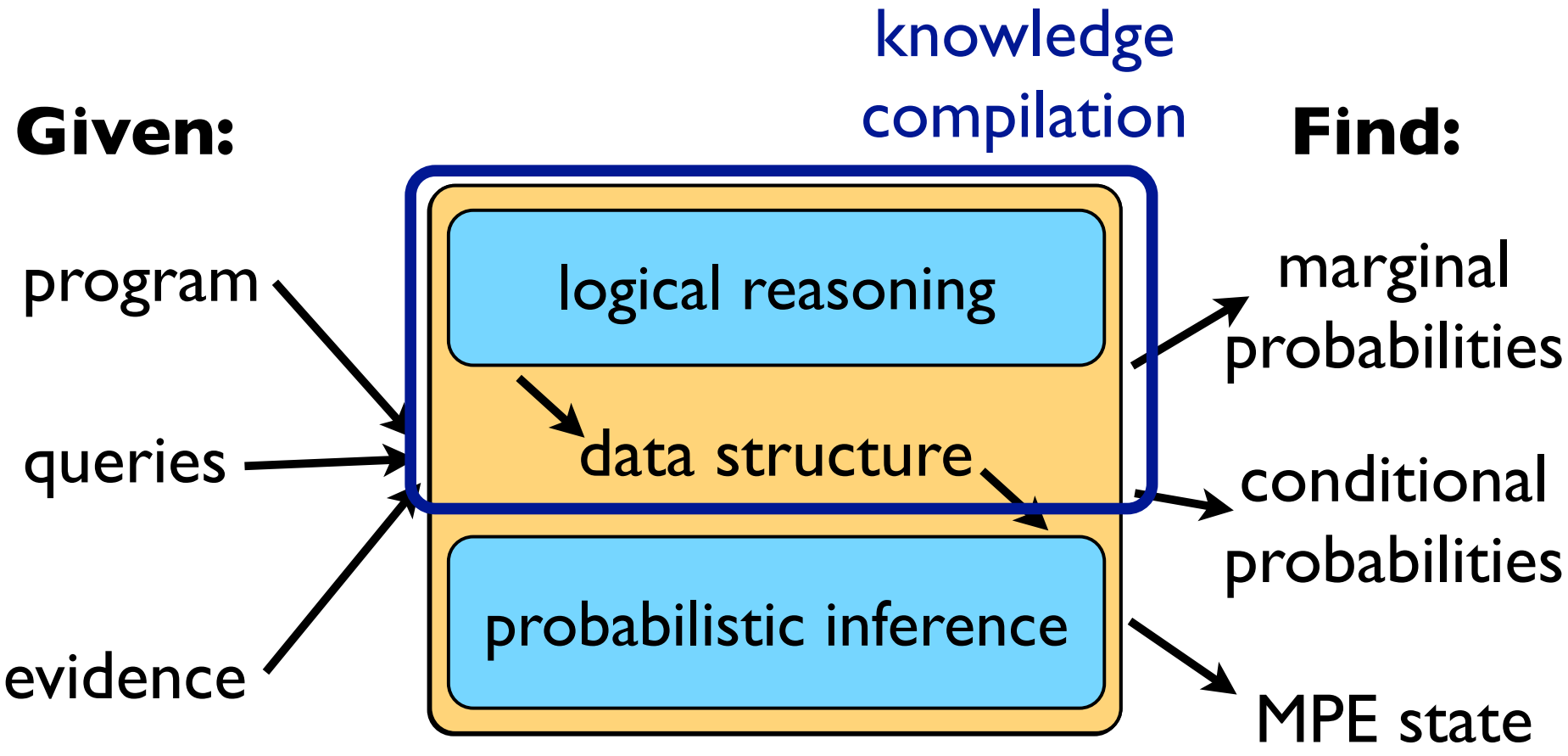
Answering Questions



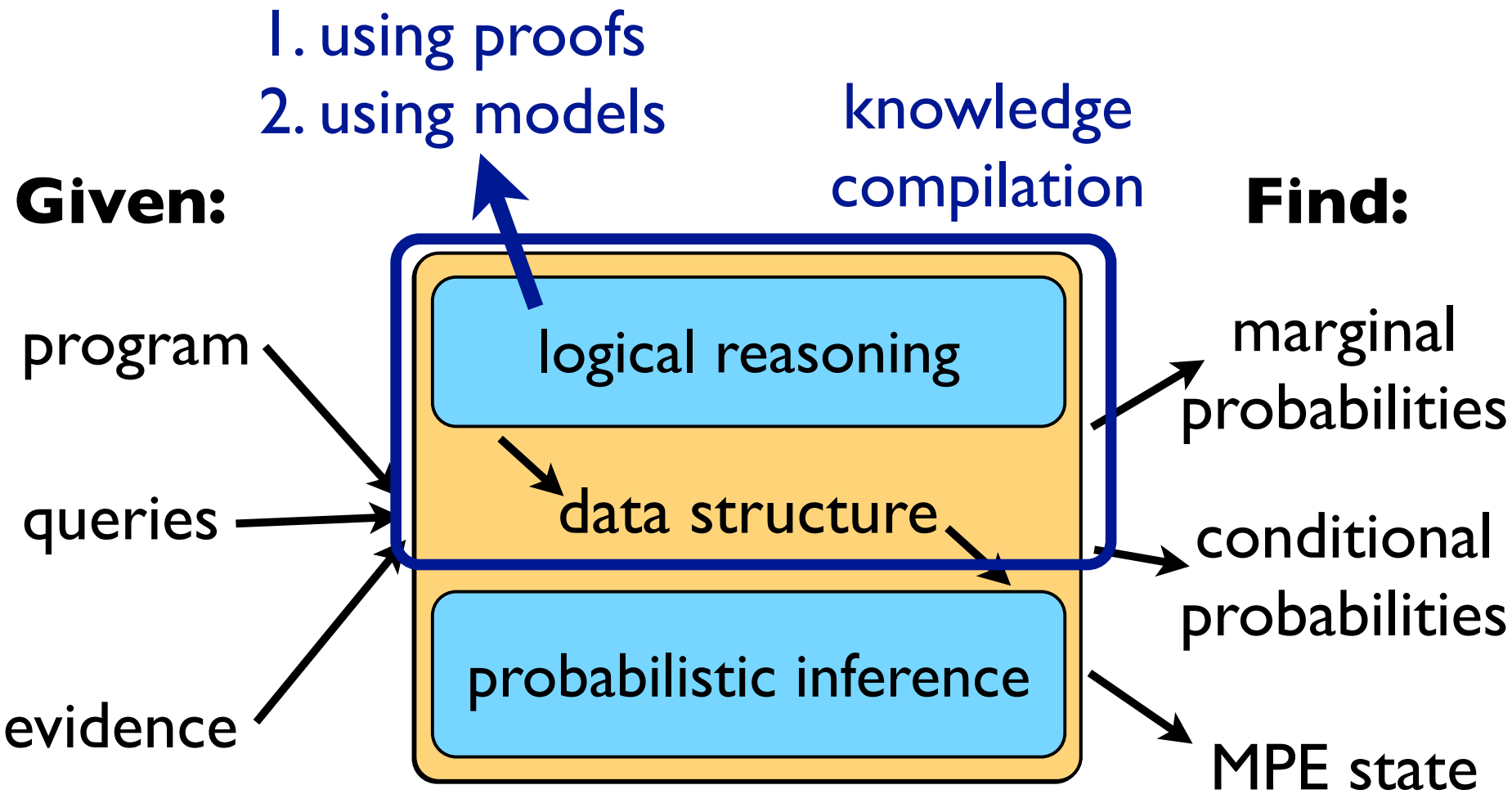
Answering Questions



Answering Questions



Answering Questions



Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

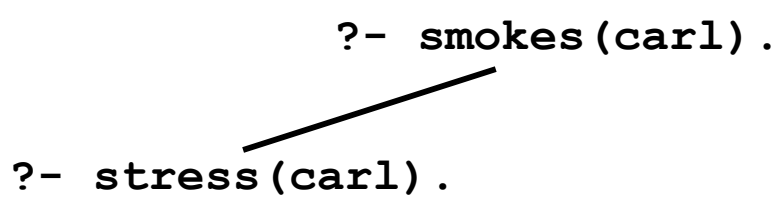
Logical Reasoning: Proofs in Prolog

```
?- smokes(carl).
```

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

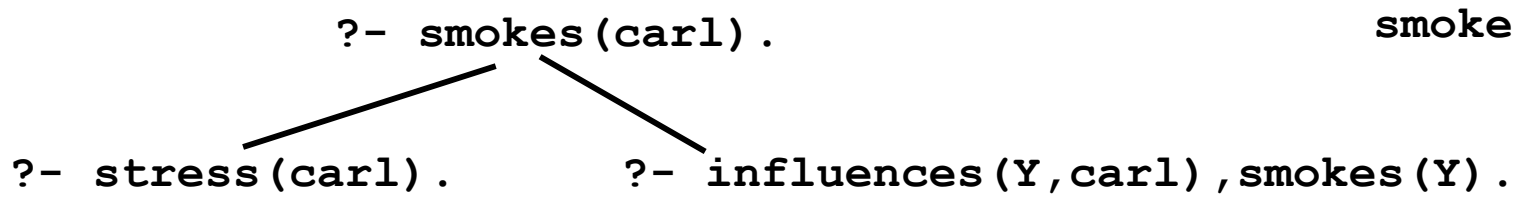
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```



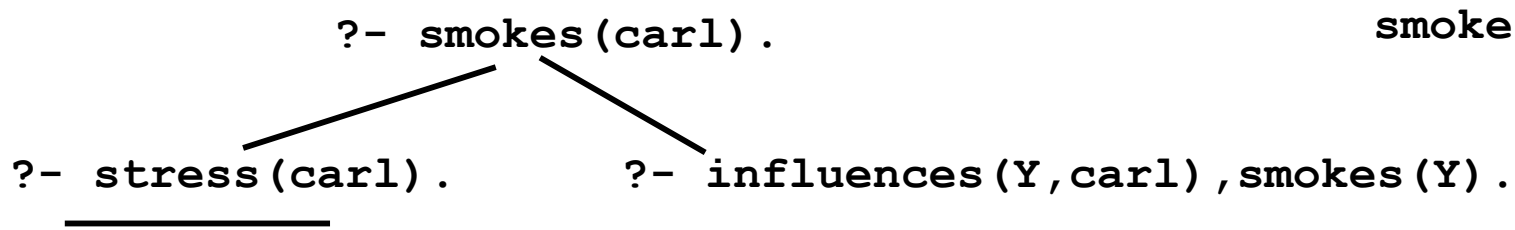
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```



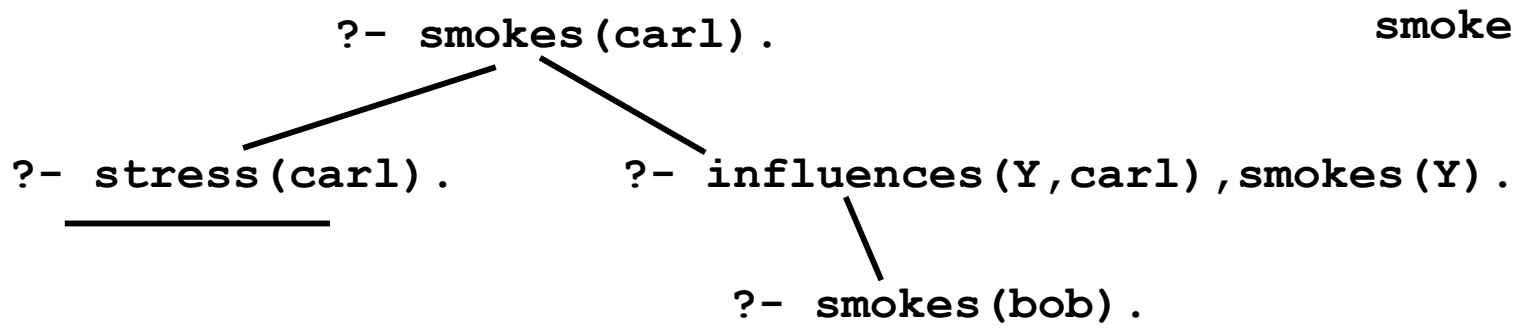
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```



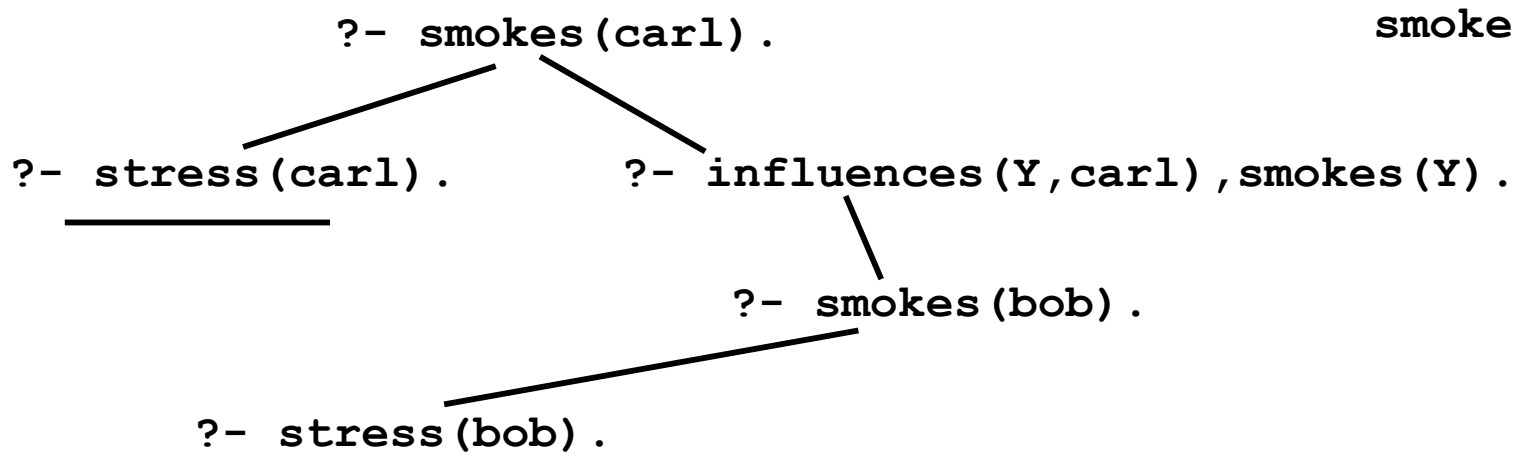
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```



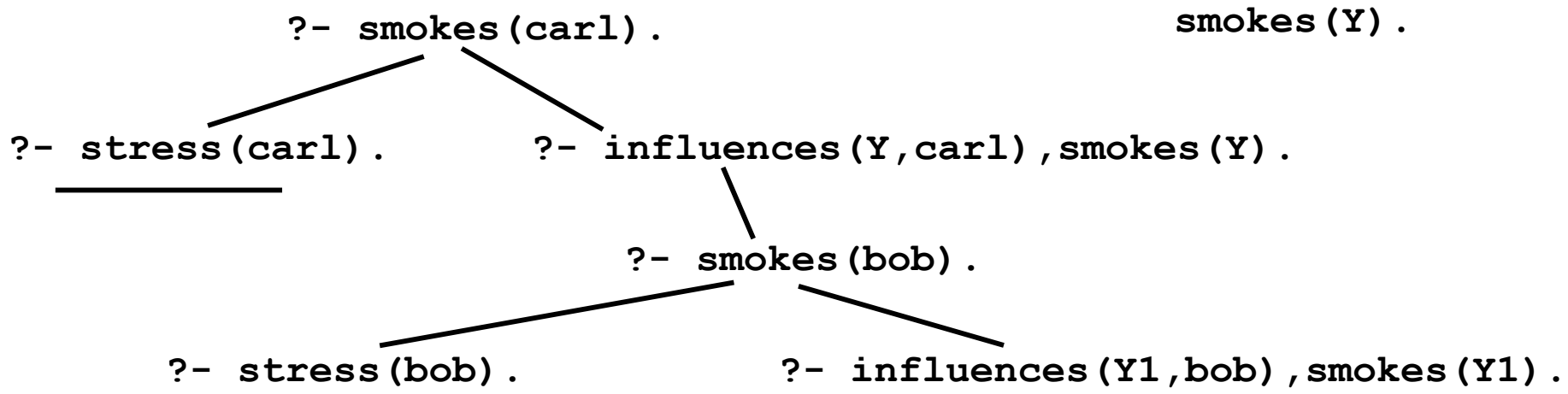
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```



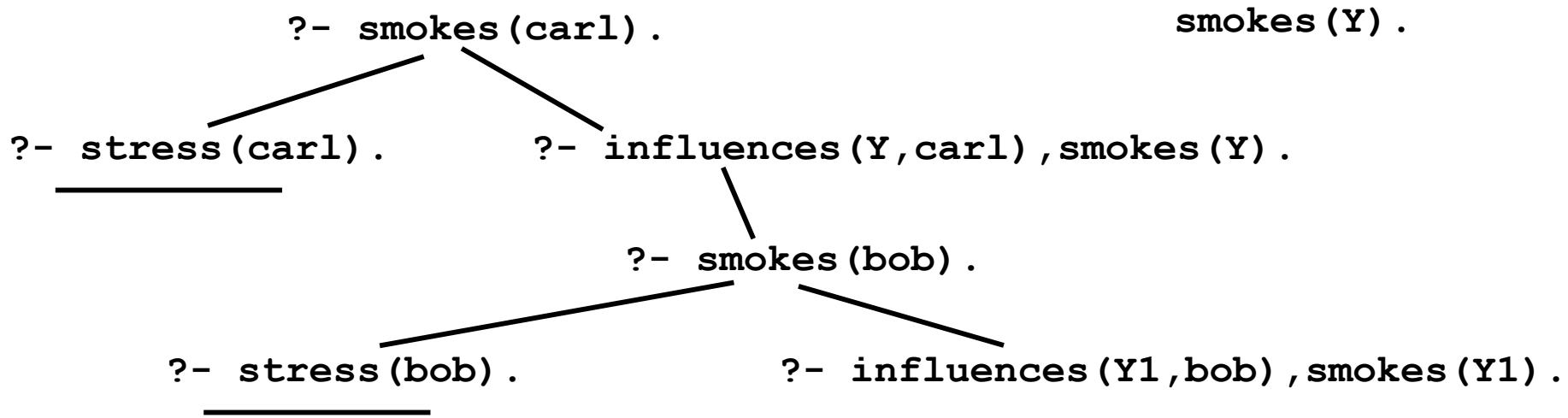
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```



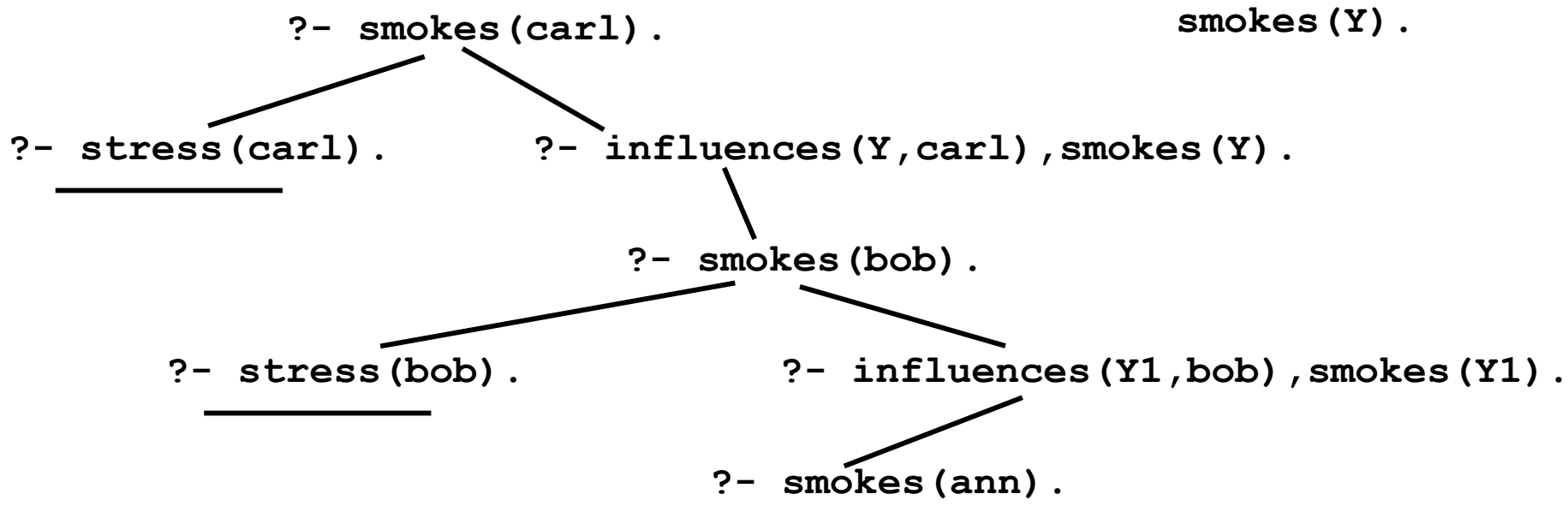
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```



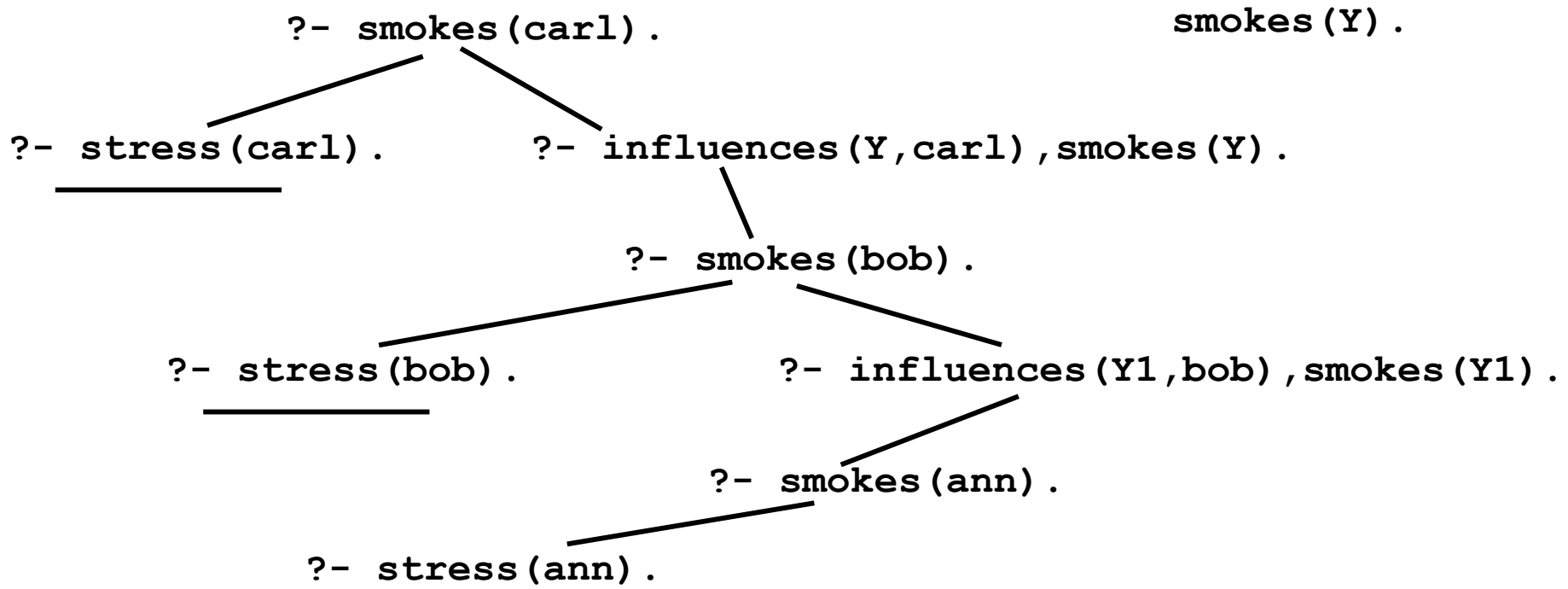
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```



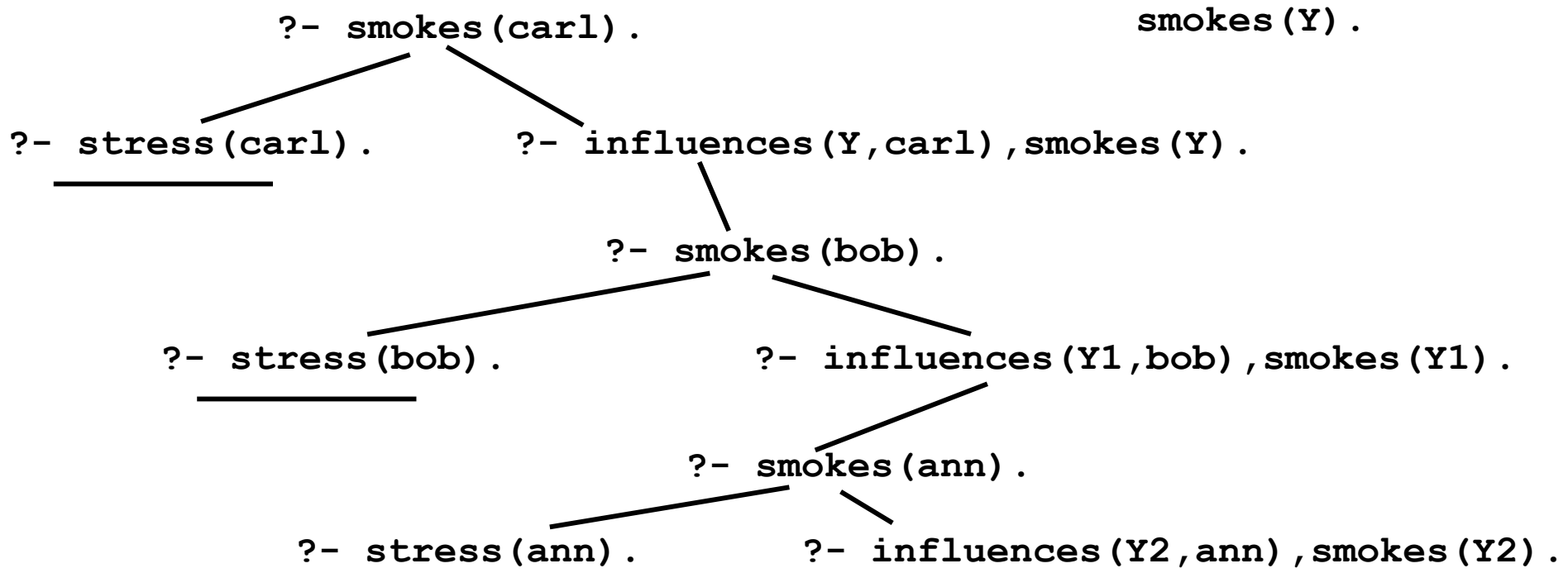
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```



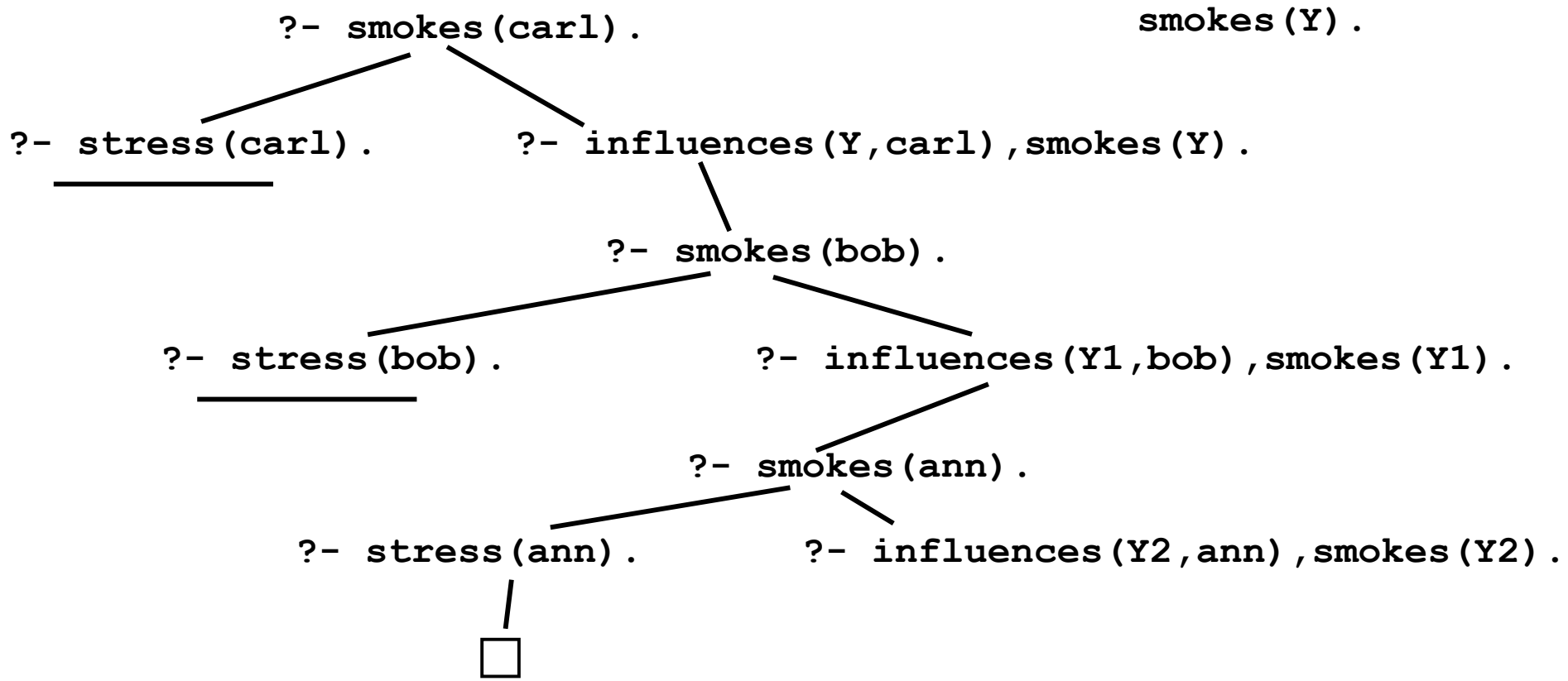
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```



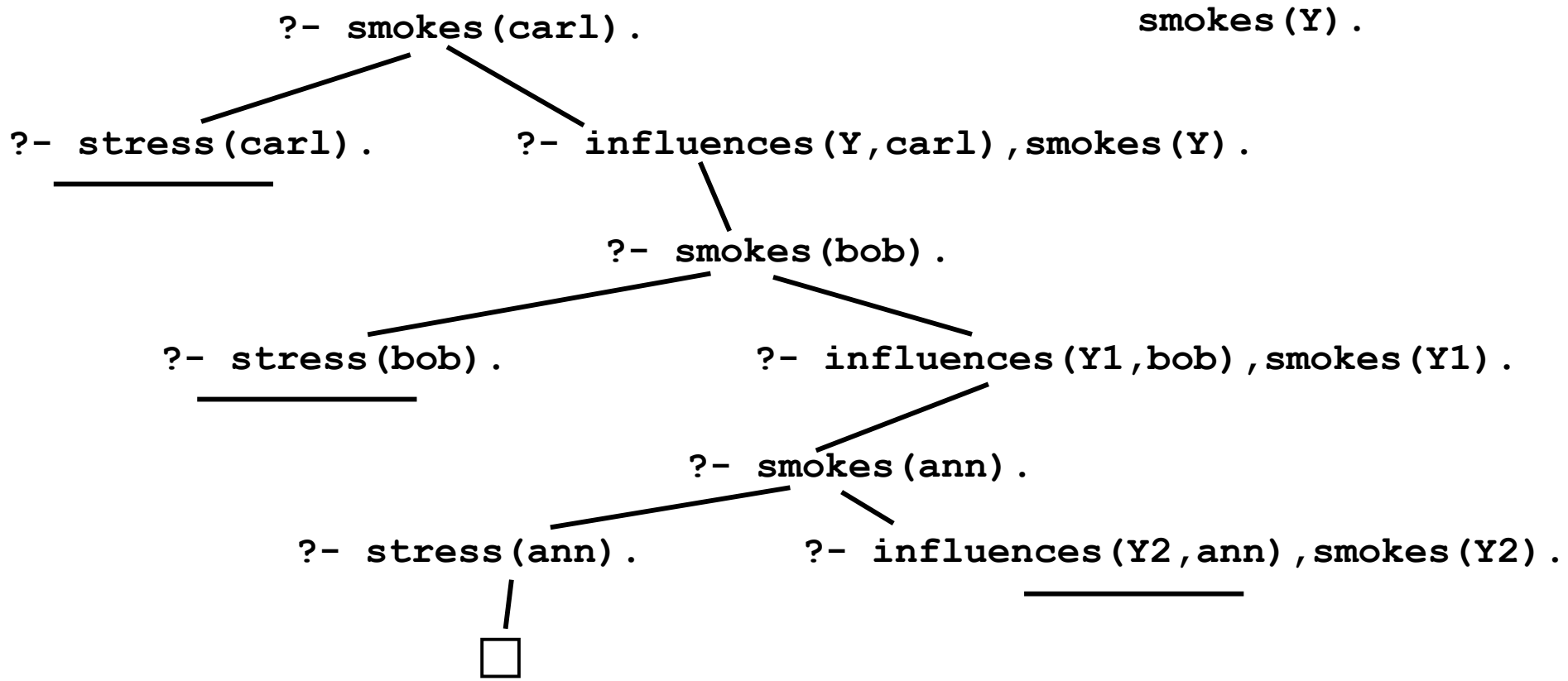
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```



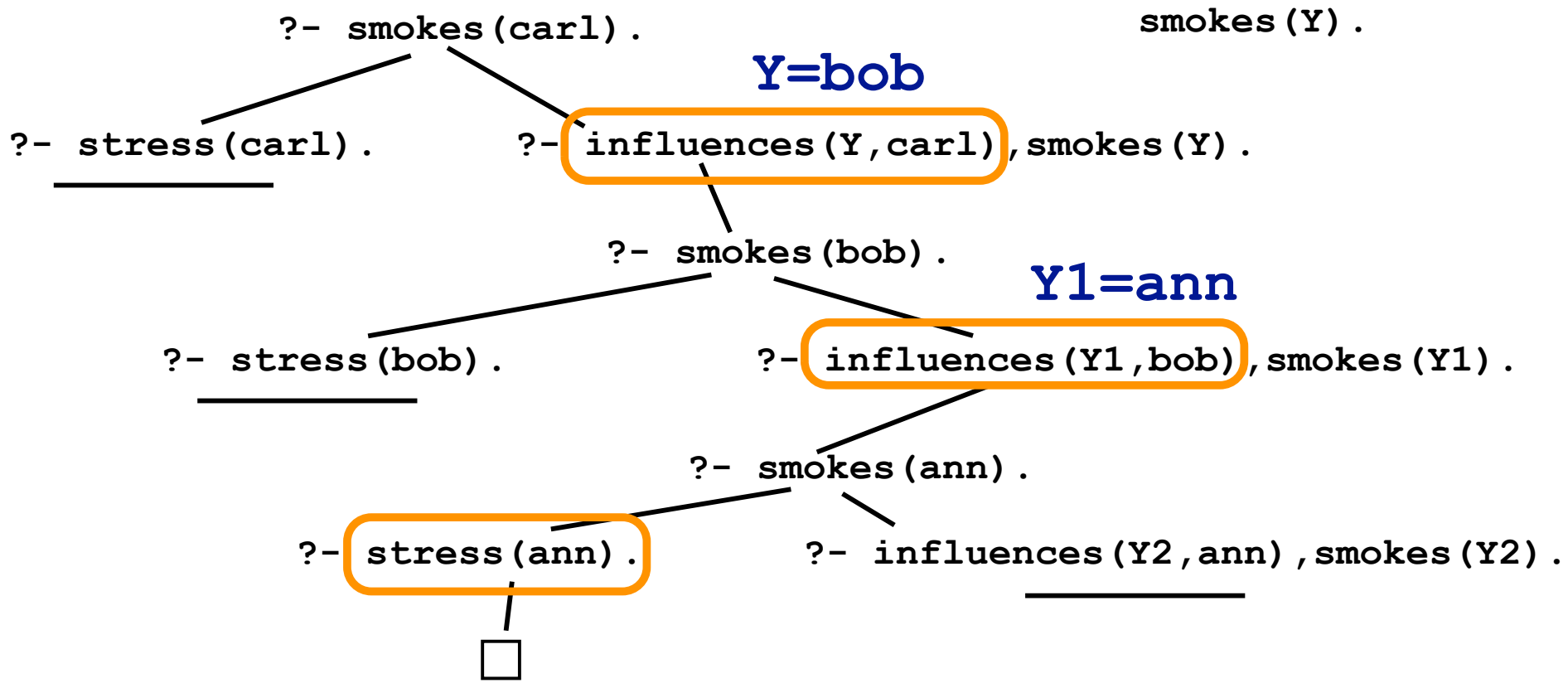
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```



Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

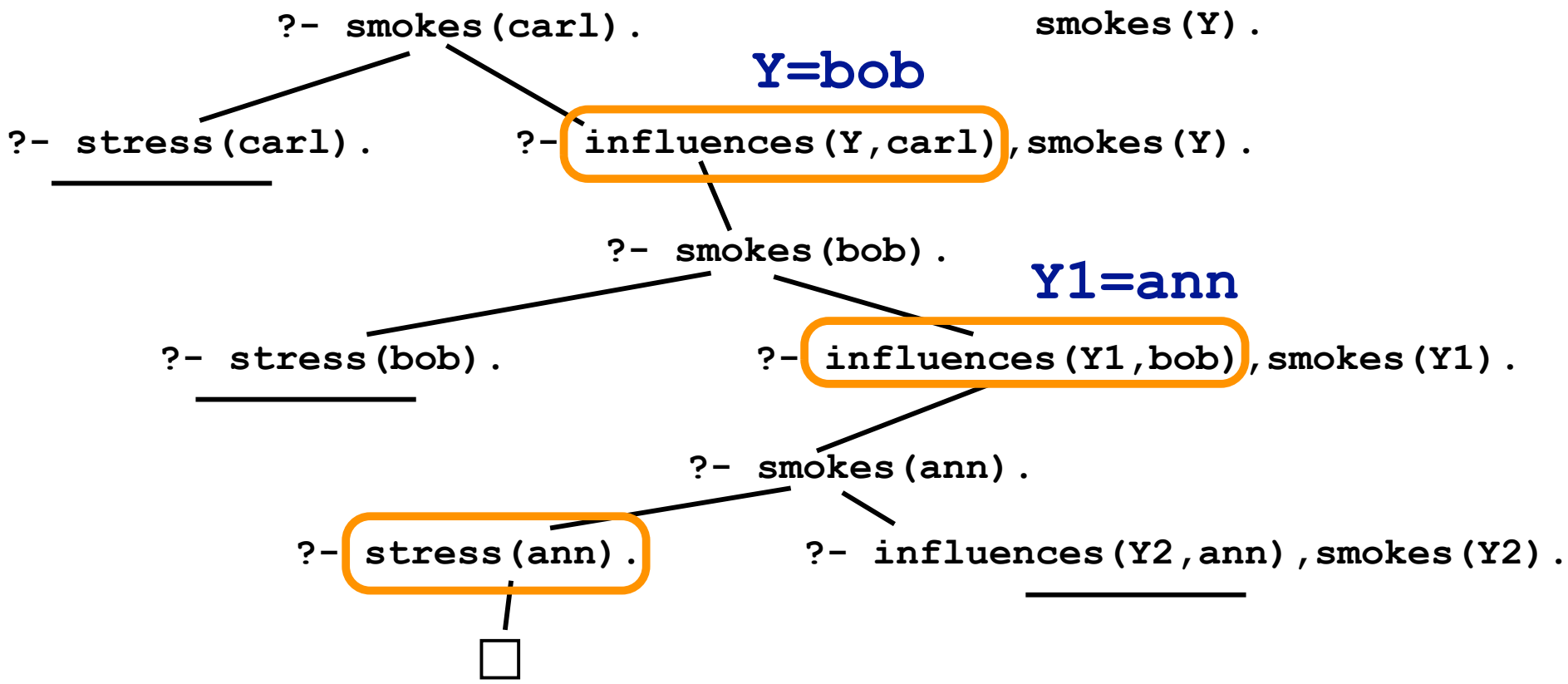


proof = facts used in successful derivation:

`influences(bob,carl) &influences(ann,bob) &stress(ann)`

Proofs in ProbLog

```
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

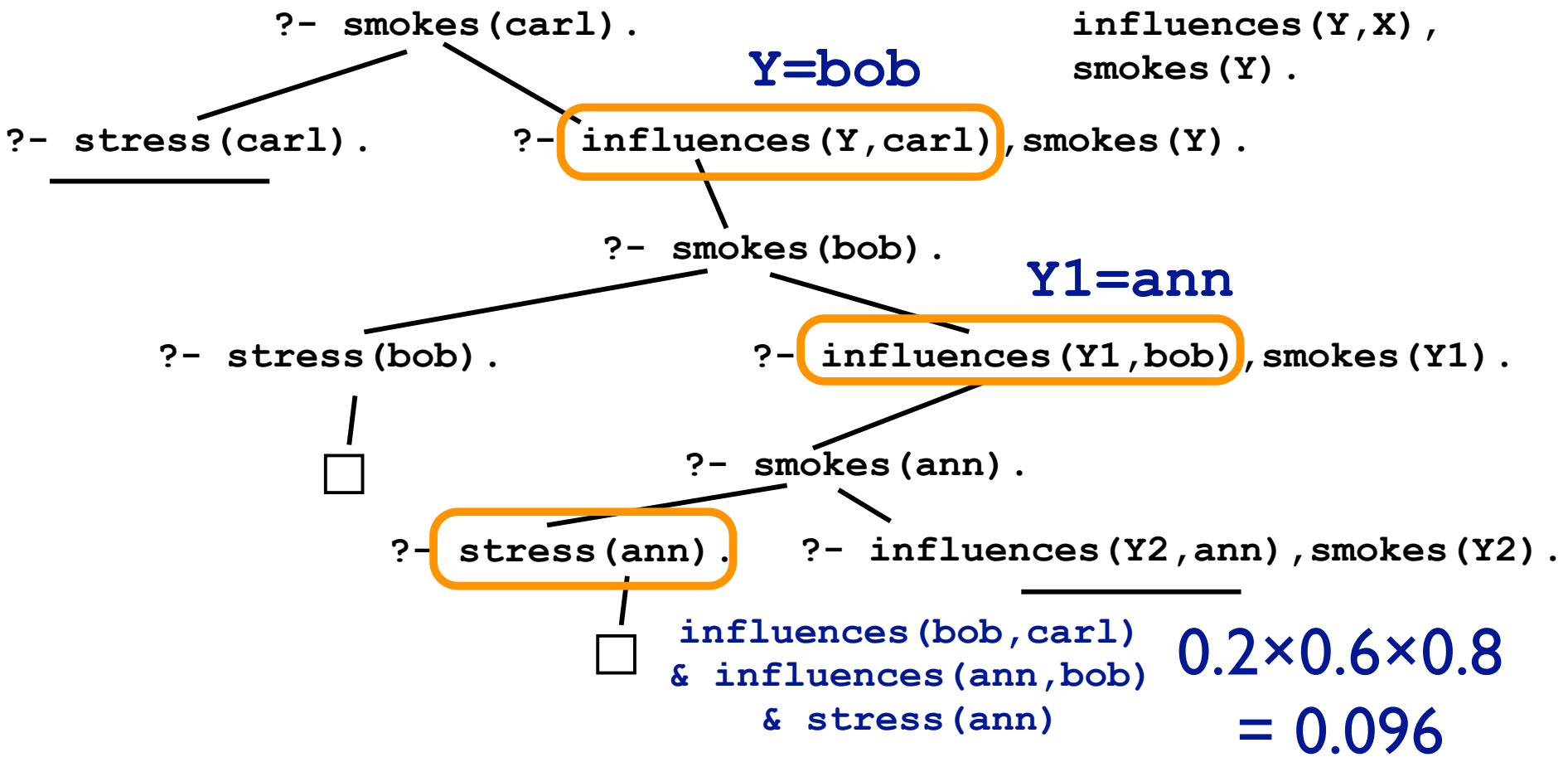


`influences(bob,carl)&influences(ann,bob)&stress(ann)`

probability of proof = 0.2 × 0.6 × 0.8 = 0.096

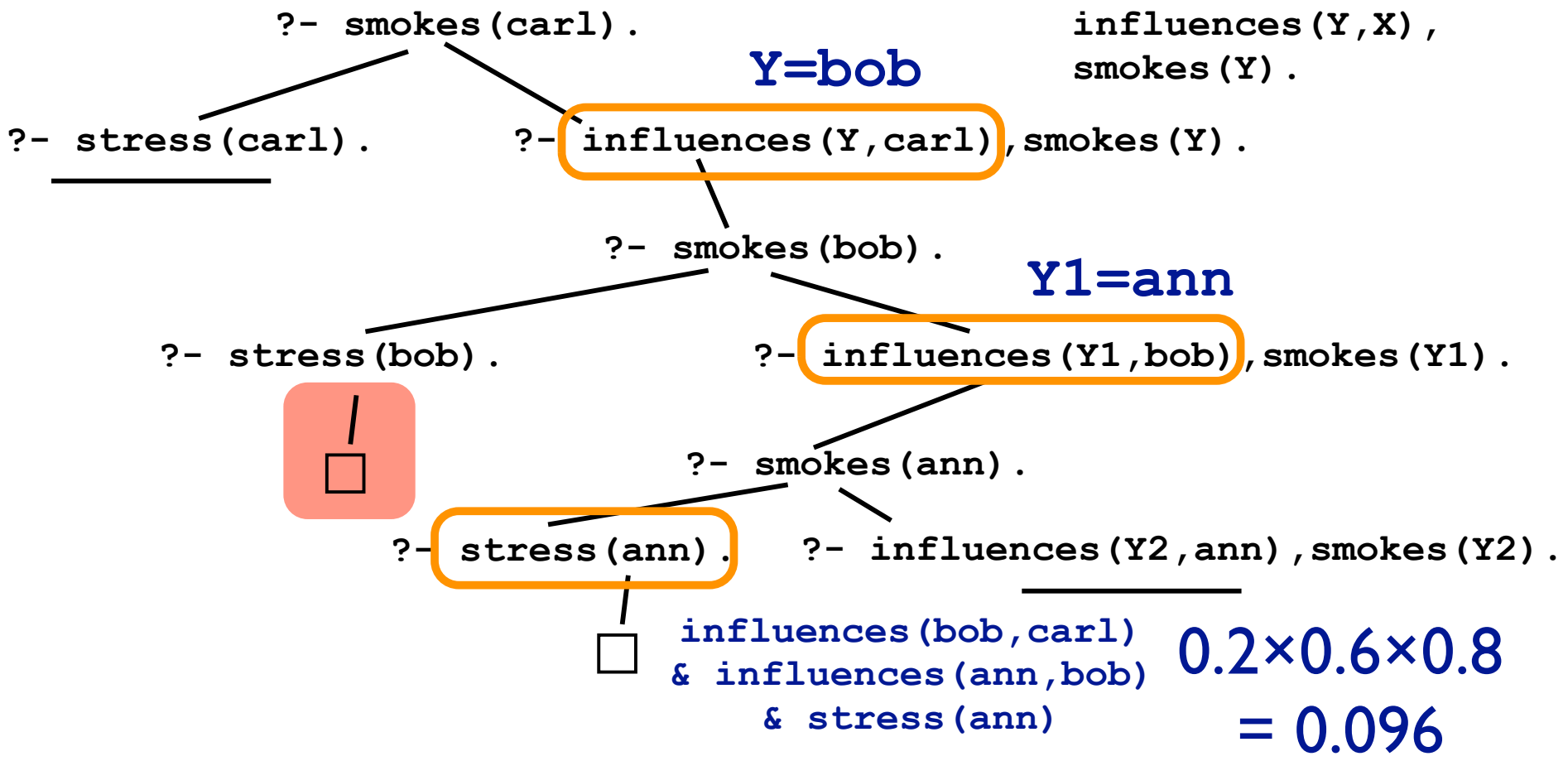
Proofs in ProbLog

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```



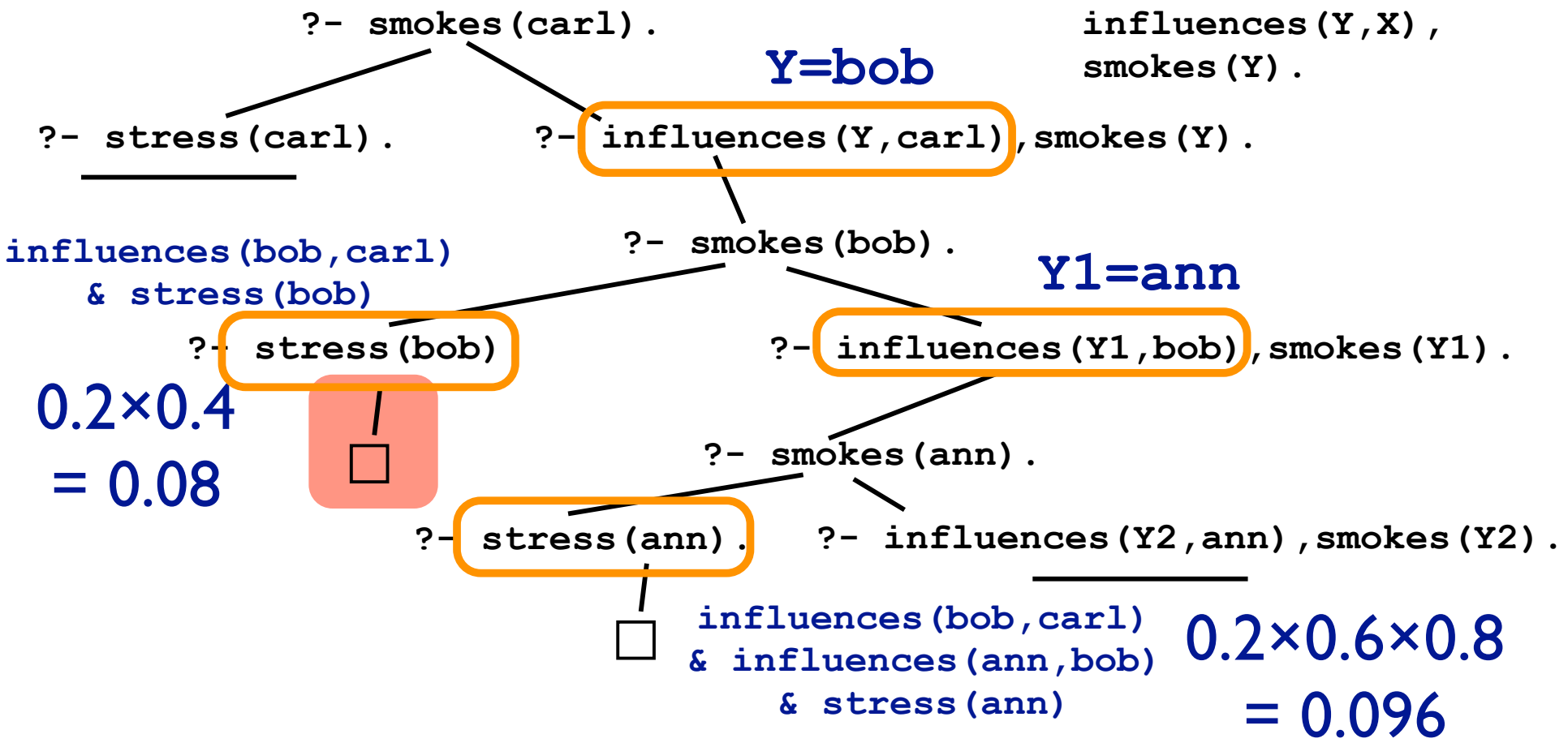
Proofs in ProbLog

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
    influences(Y,X),
    smokes(Y).
```



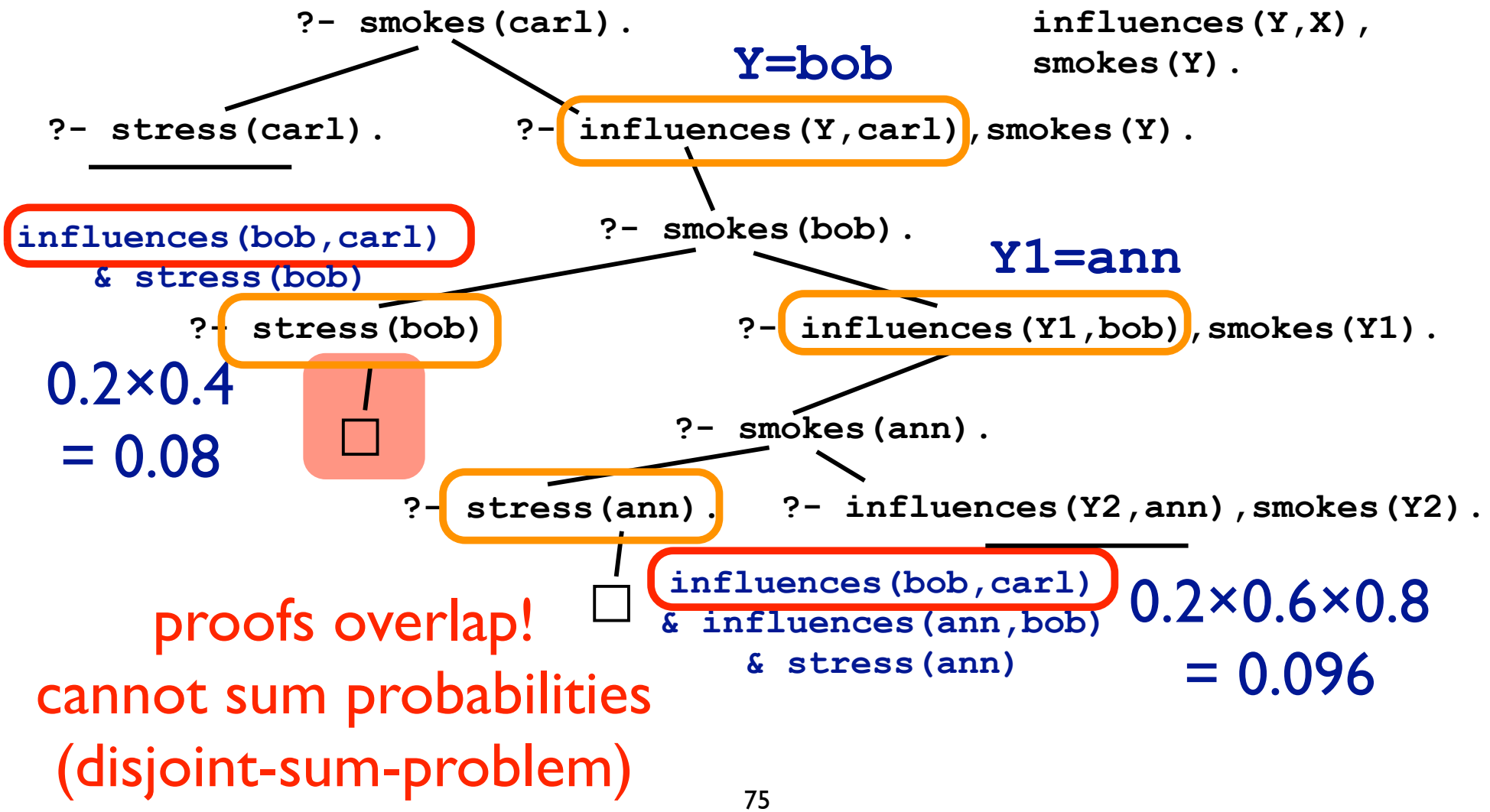
Proofs in ProbLog

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
    influences(Y,X),
    smokes(Y).
```



Proofs in ProbLog

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```



Disjoint-Sum-Problem

possible worlds

```
infl(bob,carl) &   infl(ann,bob) &   st(ann) & \+st(bob)

infl(bob,carl) &   infl(ann,bob) &   st(ann) &   st(bob)

infl(bob,carl) & \+infl(ann,bob) &   st(ann) &   st(bob)

infl(bob,carl) &   infl(ann,bob) & \+st(ann) &   st(bob)

infl(bob,carl) & \+infl(ann,bob) & \+st(ann) &   st(bob)

...
```

Disjoint-Sum-Problem

possible worlds

influences(bob,carl) &
influences(ann,bob) & stress(ann)

```
infl(bob,carl) &    infl(ann,bob) &    st(ann) & \+st(bob)

infl(bob,carl) &    infl(ann,bob) &    st(ann) &    st(bob)

infl(bob,carl) & \+infl(ann,bob) &    st(ann) &    st(bob)

infl(bob,carl) &    infl(ann,bob) & \+st(ann) &    st(bob)

infl(bob,carl) & \+infl(ann,bob) & \+st(ann) &    st(bob)

...
```

Disjoint-Sum-Problem

possible worlds

```
                                    influences(bob,carl) &
                                influences(ann,bob) & stress(ann)

infl(bob,carl) &   infl(ann,bob) &   st(ann) & \+st(bob)
infl(bob,carl) &   infl(ann,bob) &   st(ann) &   st(bob)
infl(bob,carl) & \+infl(ann,bob) &   st(ann) &   st(bob)
infl(bob,carl) &   infl(ann,bob) & \+st(ann) &   st(bob)
infl(bob,carl) & \+infl(ann,bob) & \+st(ann) &   st(bob)

...          influences(bob,carl) & stress(bob)
```

Disjoint-Sum-Problem

possible worlds

```
                                    influences(bob,carl) &
                                influences(ann,bob) & stress(ann)

infl(bob,carl) &   infl(ann,bob) &   st(ann) & \+st(bob)
infl(bob,carl) &   infl(ann,bob) &   st(ann) &   st(bob)
infl(bob,carl) & \+infl(ann,bob) &   st(ann) &   st(bob)
infl(bob,carl) &   infl(ann,bob) & \+st(ann) &   st(bob)
infl(bob,carl) & \+infl(ann,bob) & \+st(ann) &   st(bob)

...          influences(bob,carl) & stress(bob)
```

sum of proof probabilities: 0.096+0.08 = 0.1760

Disjoint-Sum-Problem

possible worlds

`influences(bob,carl) & influences(ann,bob) & stress(ann)`

`infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)`	0.0576
`infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)`	0.0384
`infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)`	0.0256
`infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)`	0.0096
`infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)`	0.0064
... `influences(bob,carl) & stress(bob)`	∑ = 0.1376

sum of proof probabilities: 0.096+0.08 = 0.1760

Disjoint-Sum-Problem

possible worlds

solution: knowledge compilation

infl(bob,carl) &	infl(ann,bob) &	st(ann) &	\+st(bob)	0.0576
infl(bob,carl) &	infl(ann,bob) &	st(ann) &	st(bob)	0.0384
infl(bob,carl) &	\+infl(ann,bob) &	st(ann) &	st(bob)	0.0256
infl(bob,carl) &	infl(ann,bob) &	\+st(ann) &	st(bob)	0.0096
infl(bob,carl) &	\+infl(ann,bob) &	\+st(ann) &	st(bob)	0.0064

... influences(bob,carl) & stress(bob)

$\sum = 0.1376$

sum of proof probabilities: 0.096+0.08 = 0.1760

Binary Decision Diagrams [Bryant 86]

- compact graphical representation of Boolean formula
- popular in many branches of CS
- automatically disjoins proofs
  → efficient probability computation

Binary Decision Diagrams

[Bryant 86]

$X \vee Y \vee Z$

Binary Decision Diagrams

[Bryant 86]

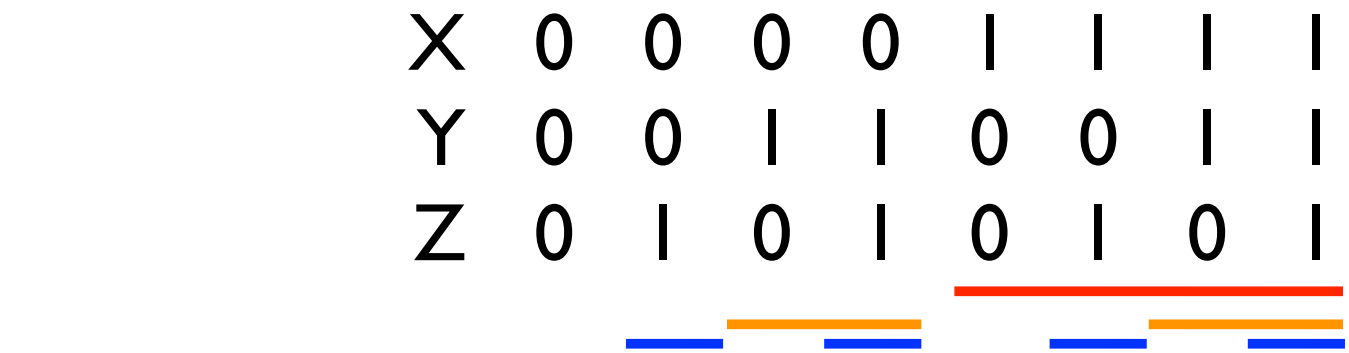
$X \vee Y \vee Z$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]

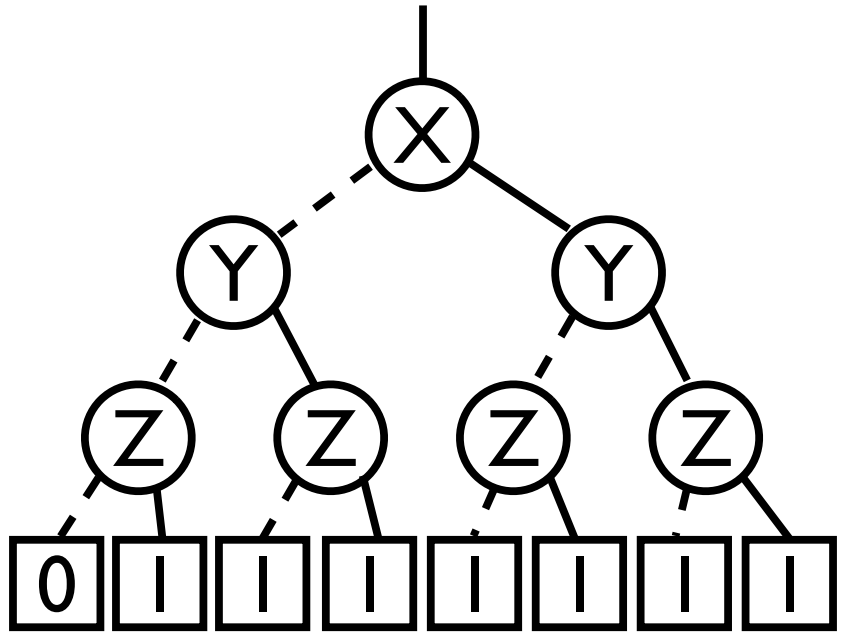
$X \vee Y \vee Z$



Binary Decision Diagrams

[Bryant 86]

$X \vee Y \vee Z$

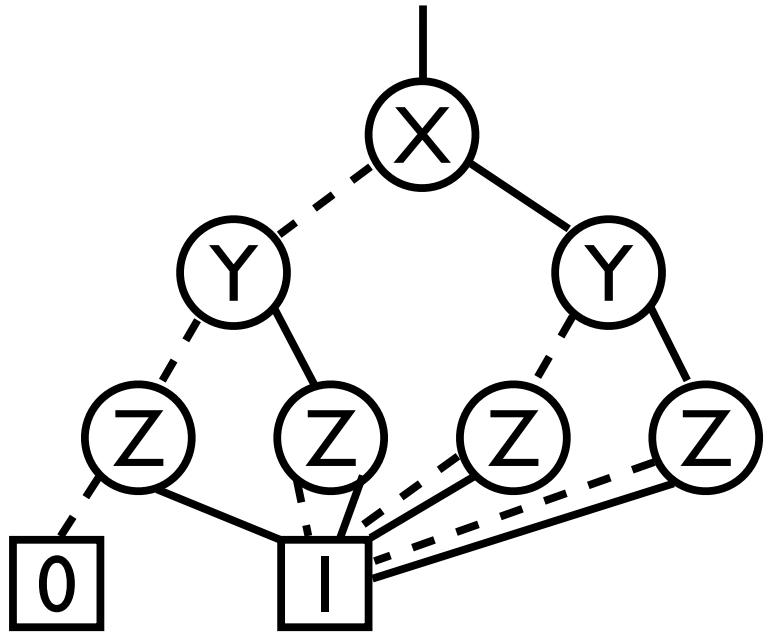


X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]

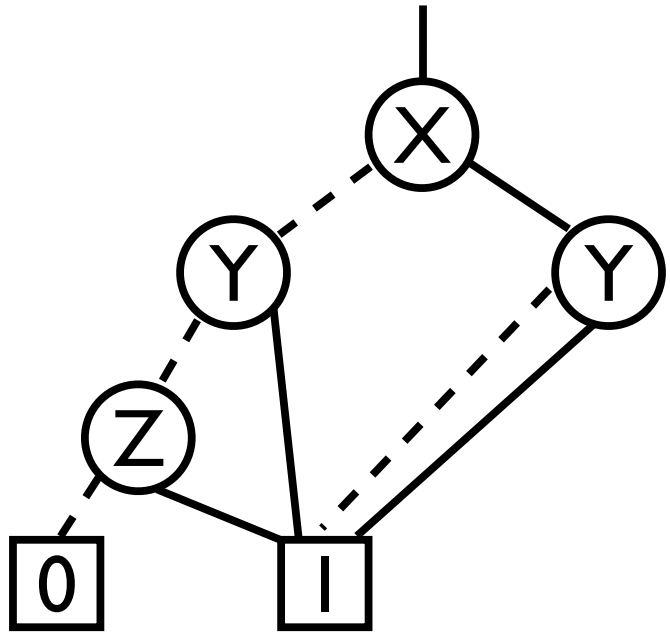
$X \vee Y \vee Z$



X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]

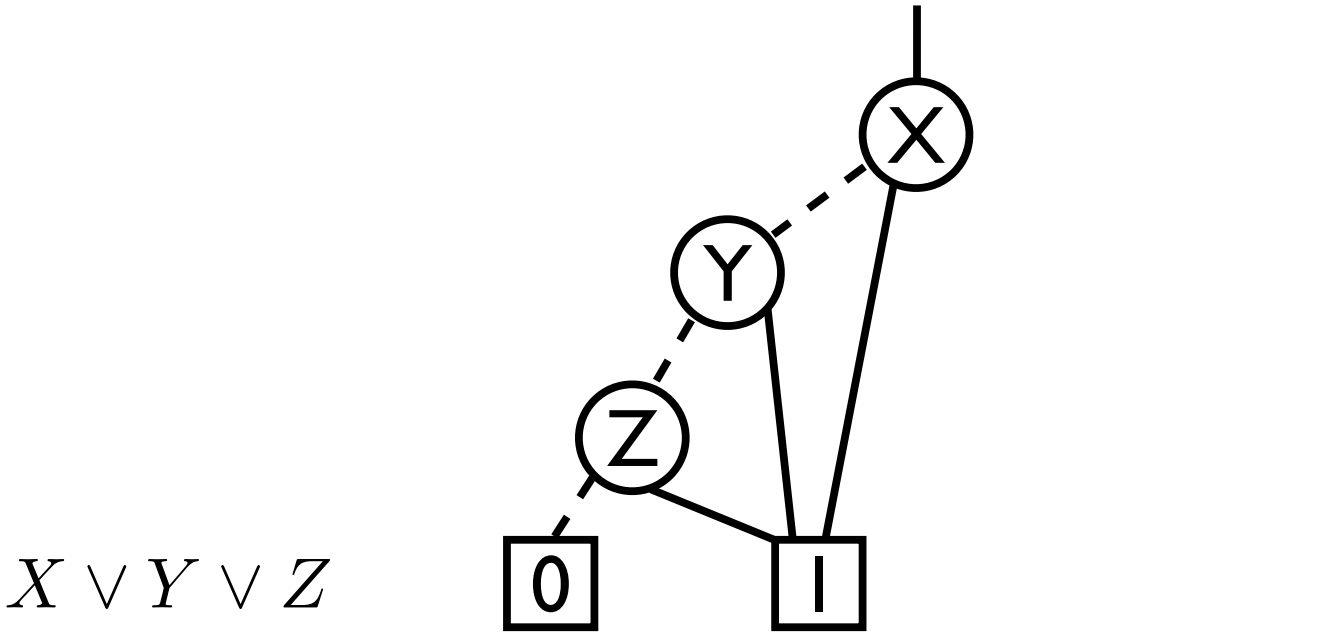


$X \vee Y \vee Z$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]

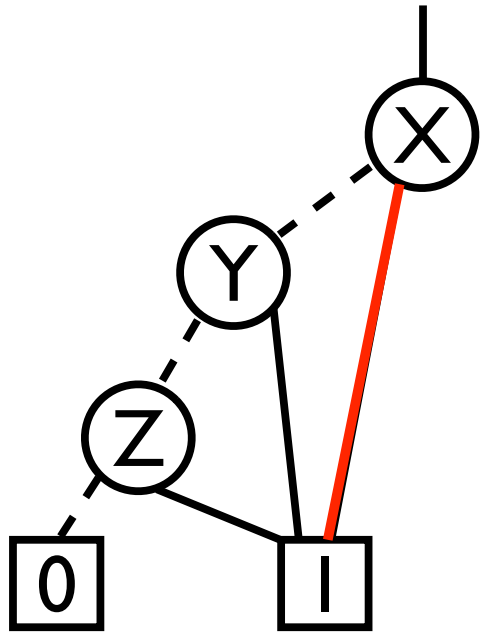


$X \vee Y \vee Z$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]



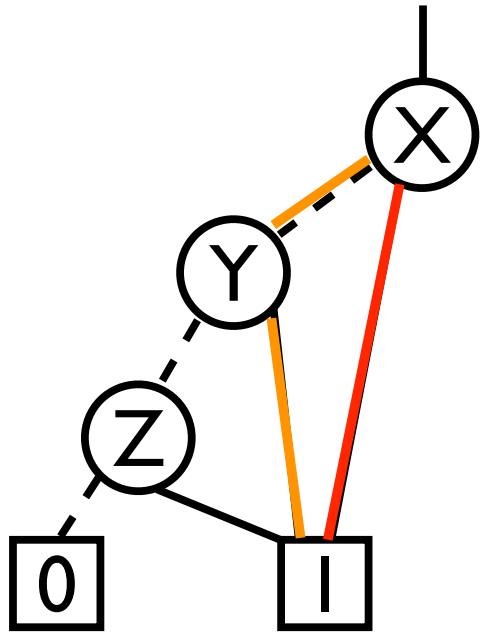
$X \vee Y \vee Z$

$X \vee (\neg X \wedge Y) \vee$
$(\neg X \wedge \neg Y \wedge Z)$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]



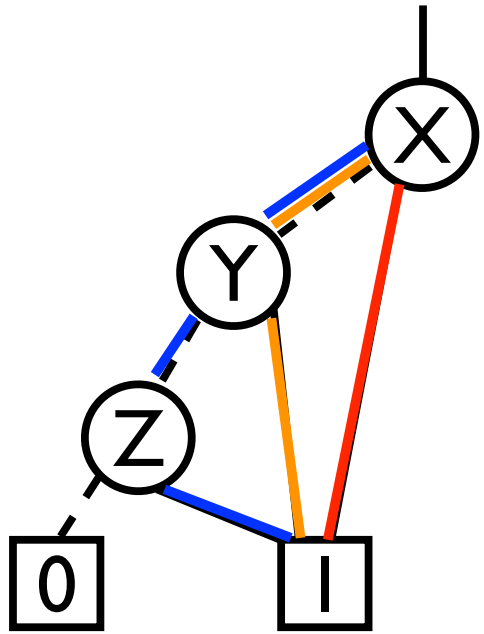
$X \vee Y \vee Z$

$X \vee (\neg X \wedge Y) \vee (\neg X \wedge \neg Y \wedge Z)$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]



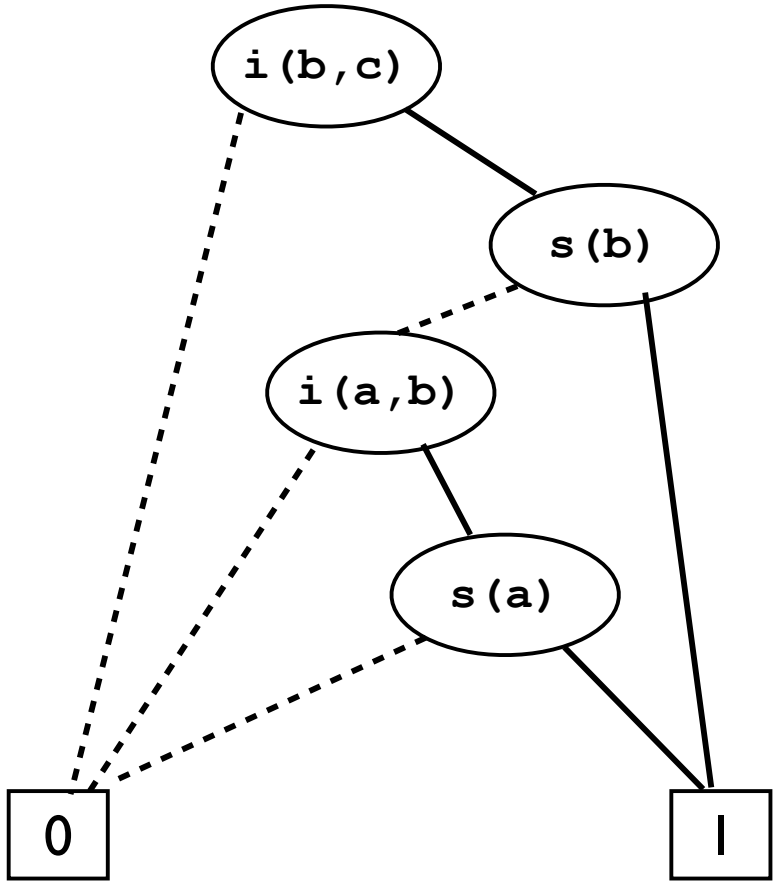
$X \vee Y \vee Z$

$X \vee (\neg X \wedge Y) \vee (\neg X \wedge \neg Y \wedge Z)$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

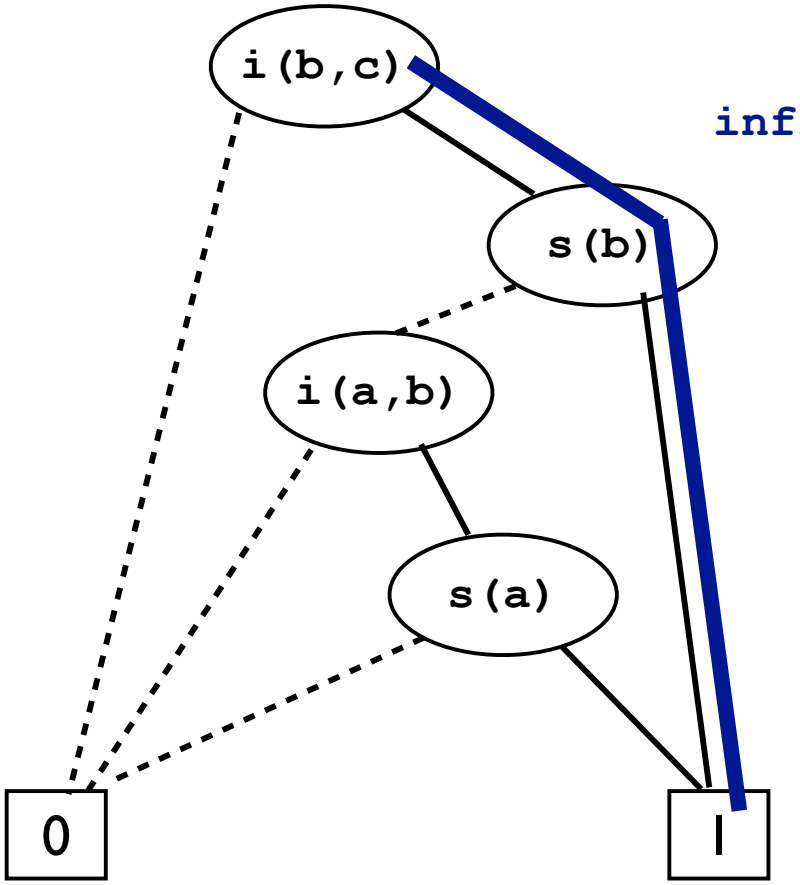
Binary Decision Diagrams

[Bryant 86]



Binary Decision Diagrams

[Bryant 86]



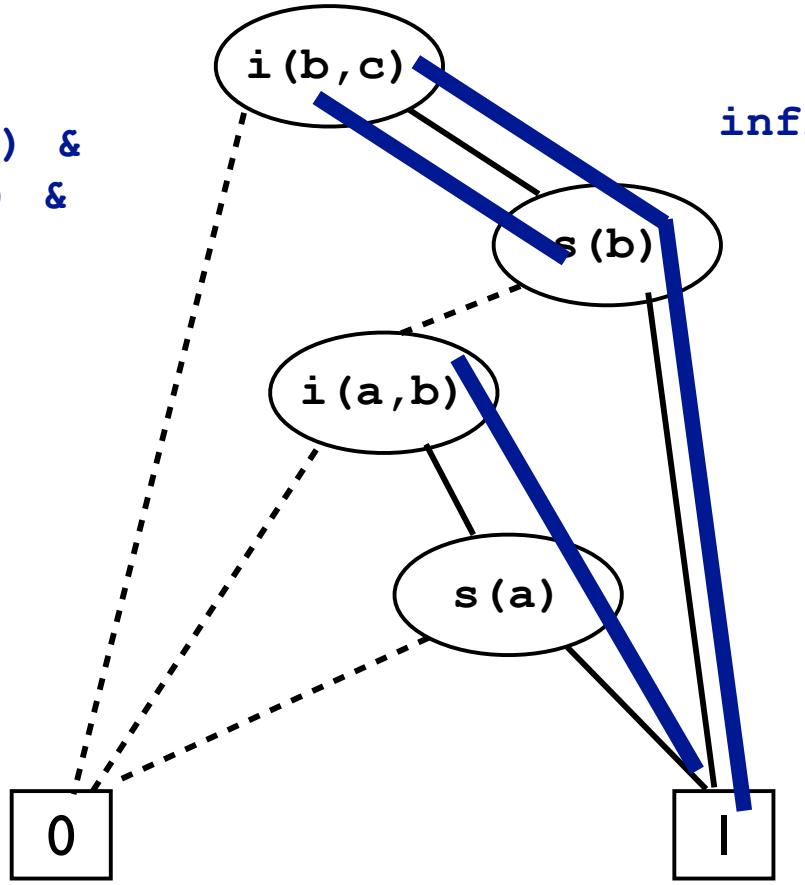
`influences(bob,carl) & stress(bob)`

Binary Decision Diagrams

[Bryant 86]

influences(bob,carl) &
influences(ann,bob) &
stress(ann)

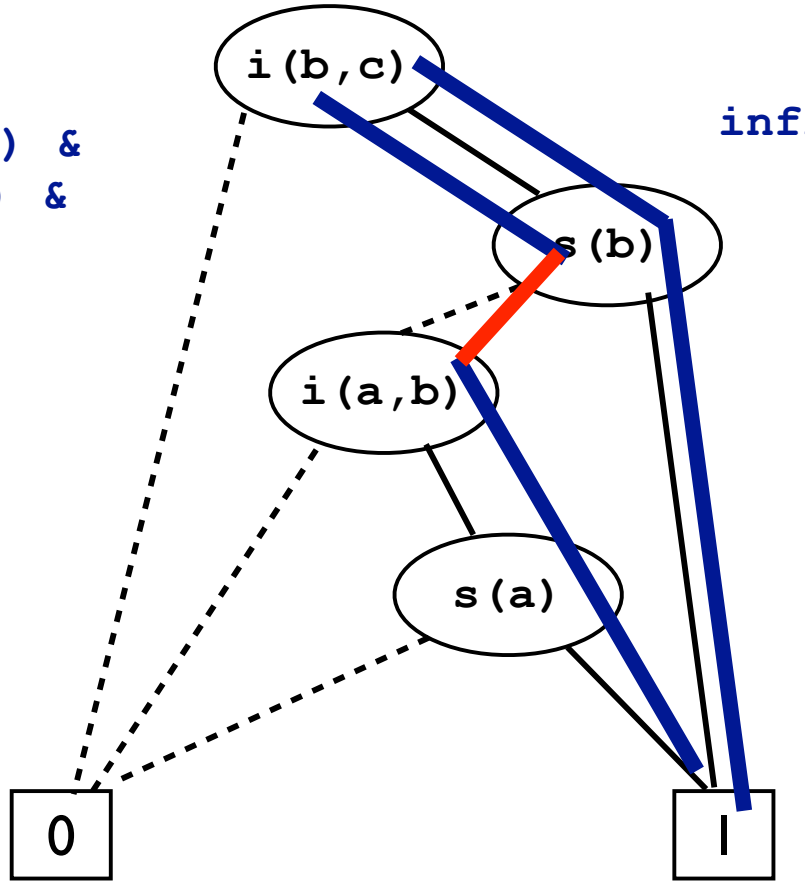
influences(bob,carl) &
stress(bob)



Binary Decision Diagrams

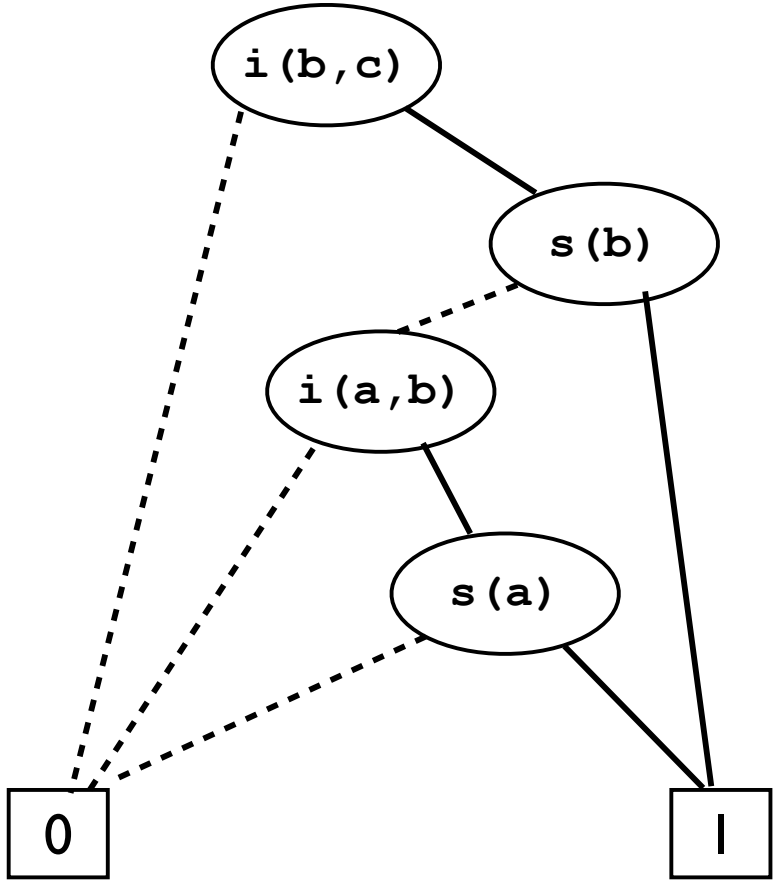
[Bryant 86]

influences(bob,carl) &
influences(ann,bob) &
stress(ann)
& not stress(bob)

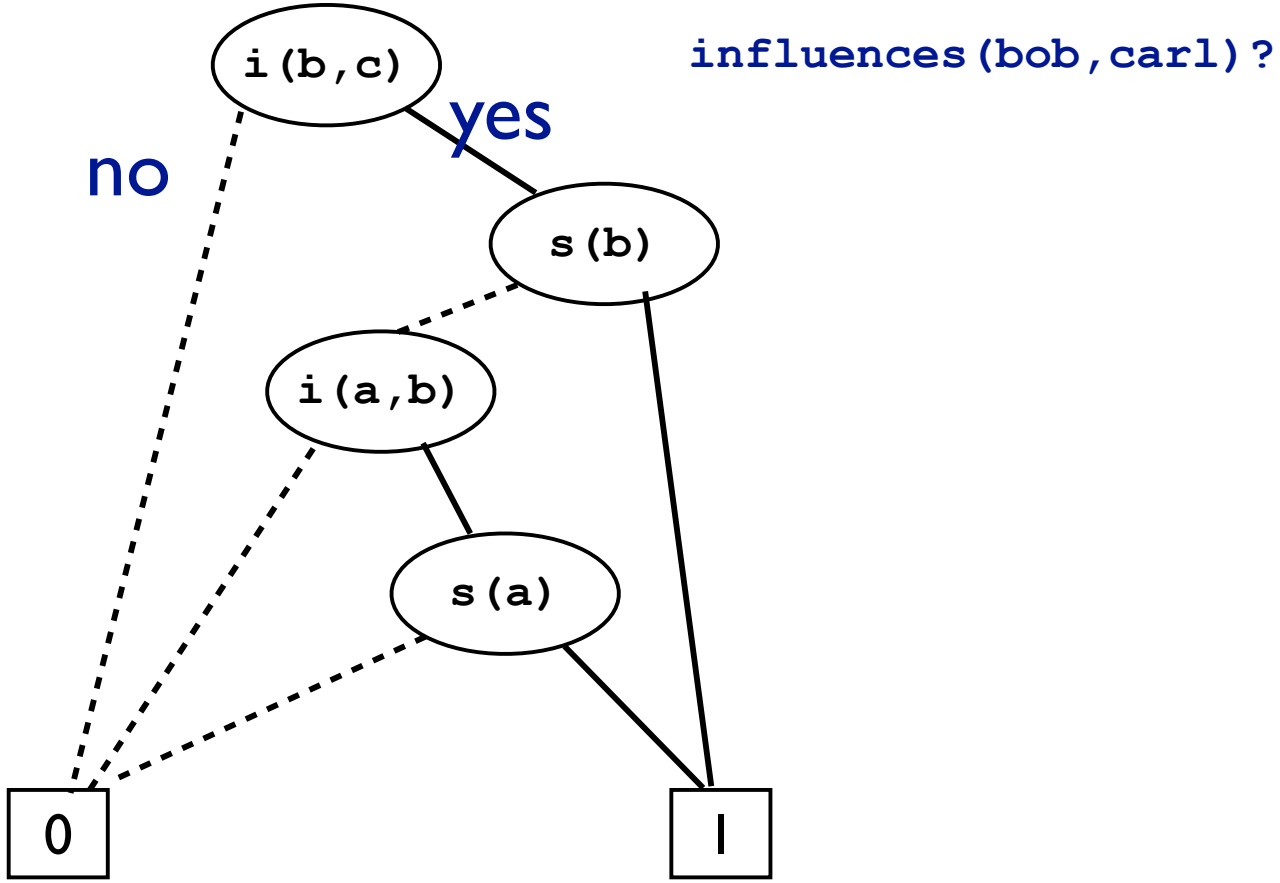


influences(bob,carl) &
stress(bob)

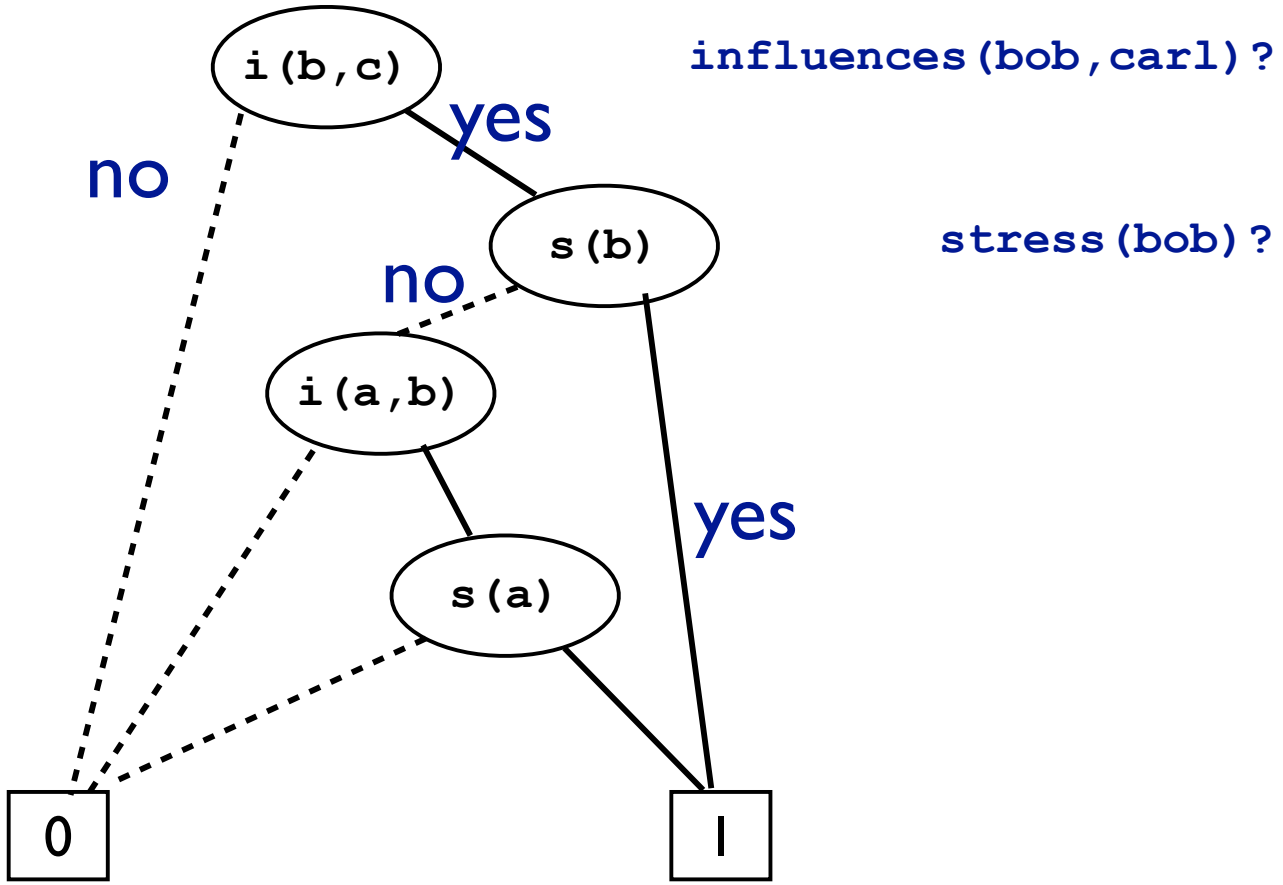
Binary Decision Diagrams



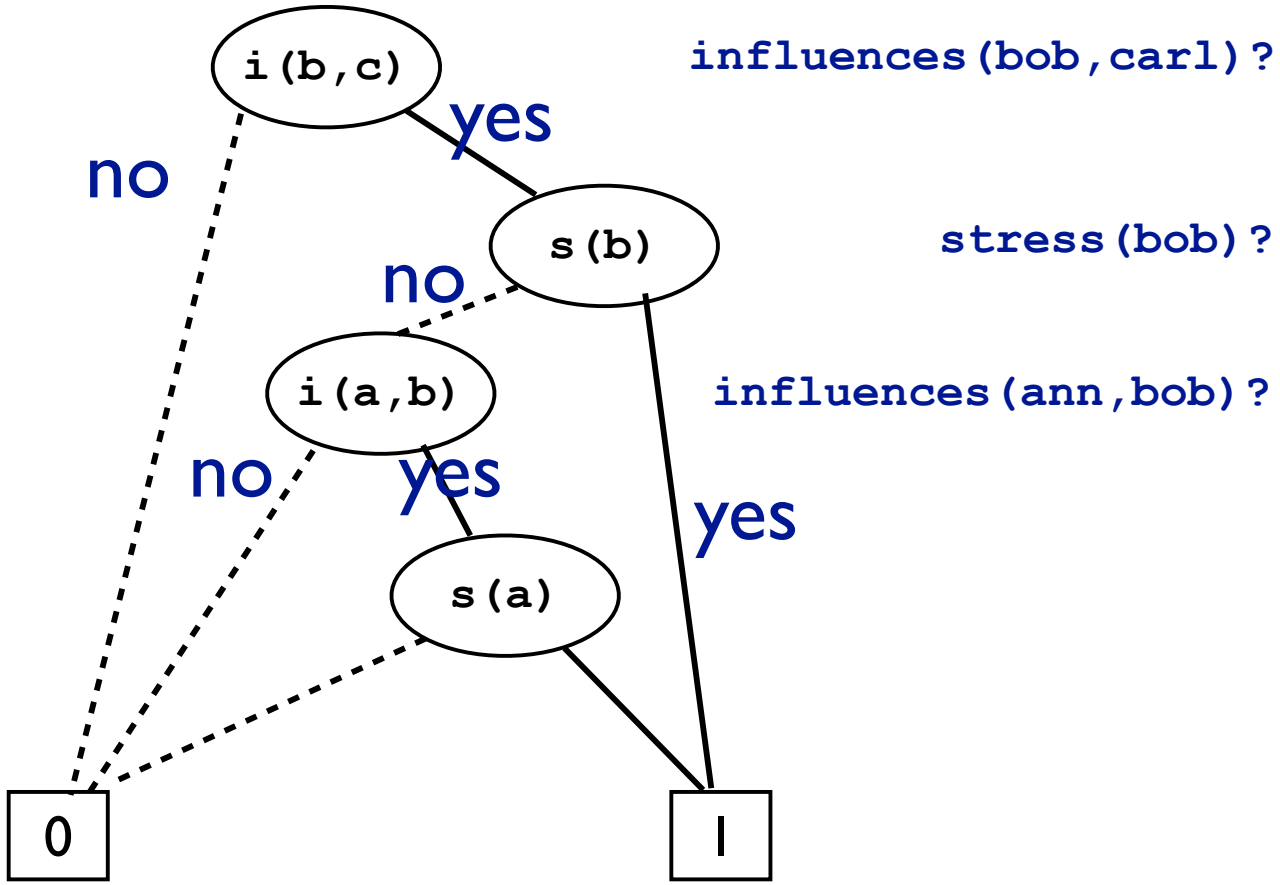
Binary Decision Diagrams



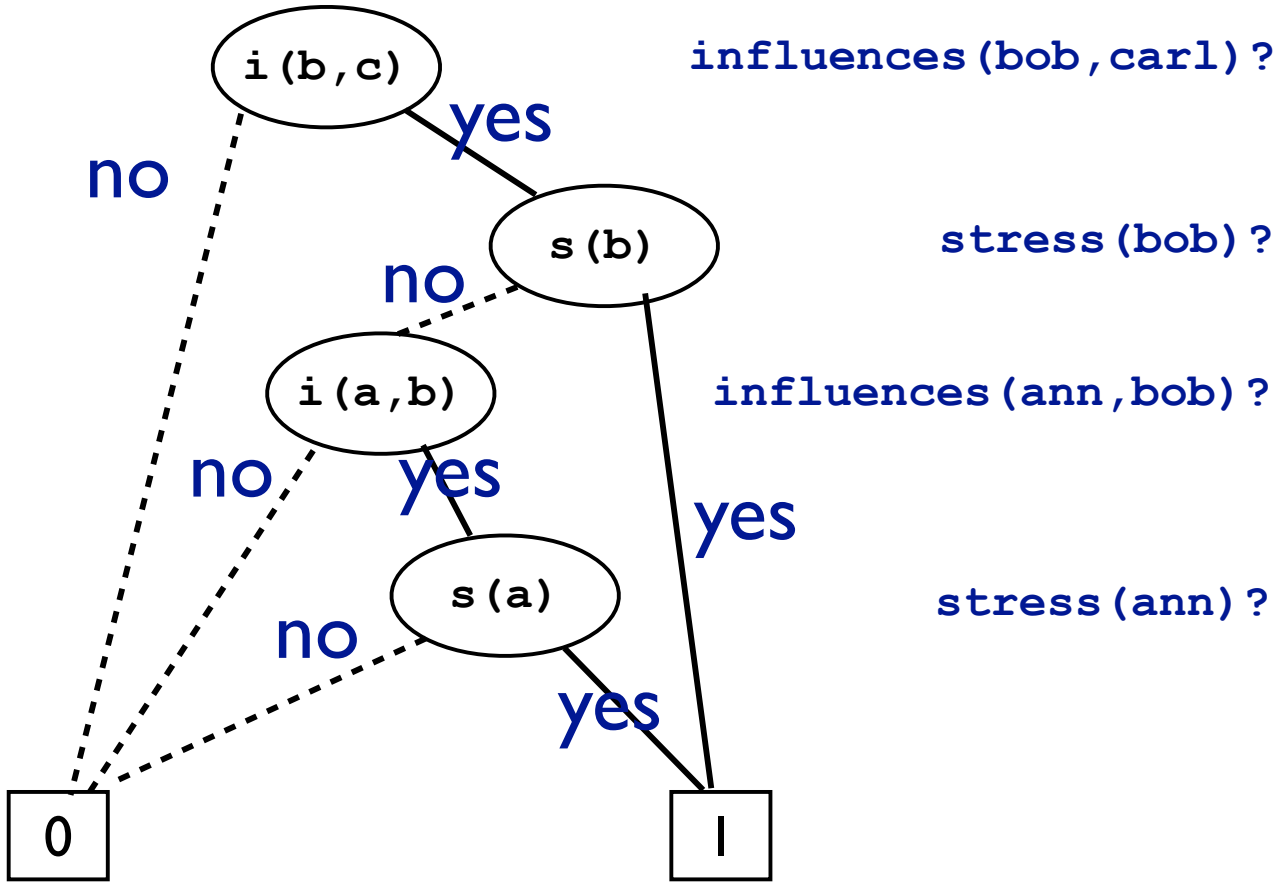
Binary Decision Diagrams



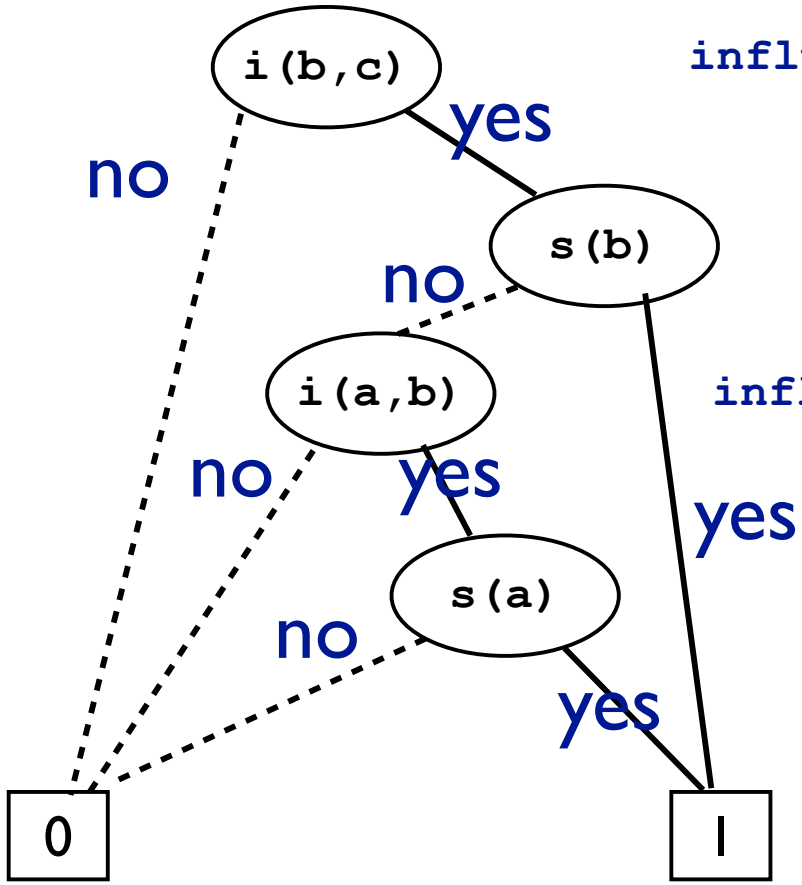
Binary Decision Diagrams



Binary Decision Diagrams



Binary Decision Diagrams



influences(bob,carl)?

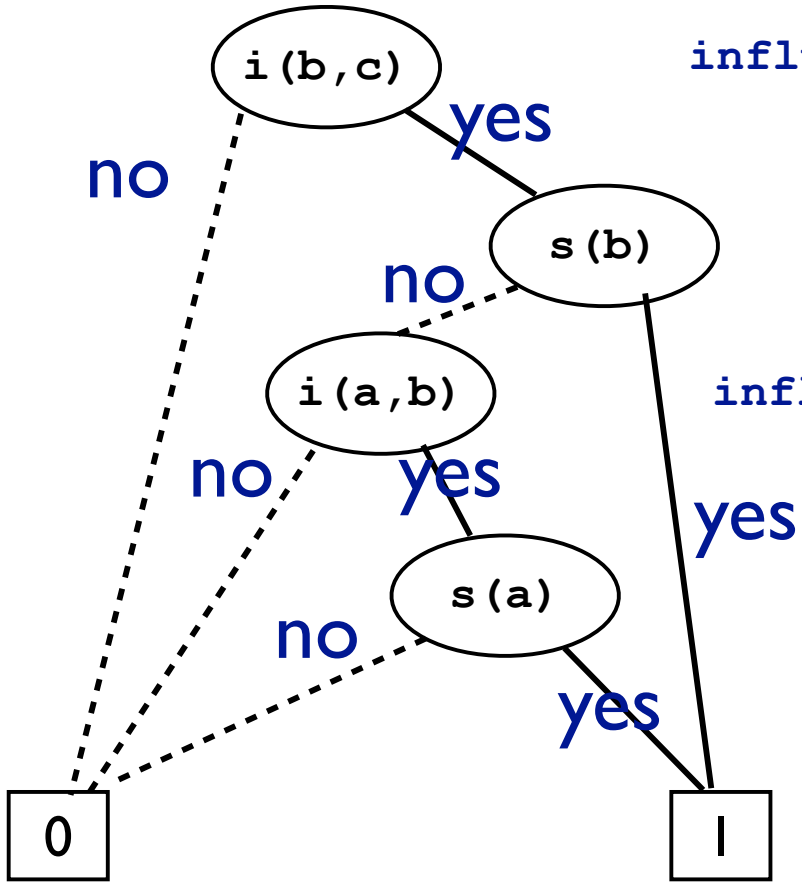
stress(bob)?

influences(ann,bob)?

stress(ann)?

smokes(c) = i(b,c)&s(b) v
i(b,c)&i(a,b)&s(a)

Binary Decision Diagrams



influences(bob,carl)?

stress(bob)?

influences(ann,bob)?

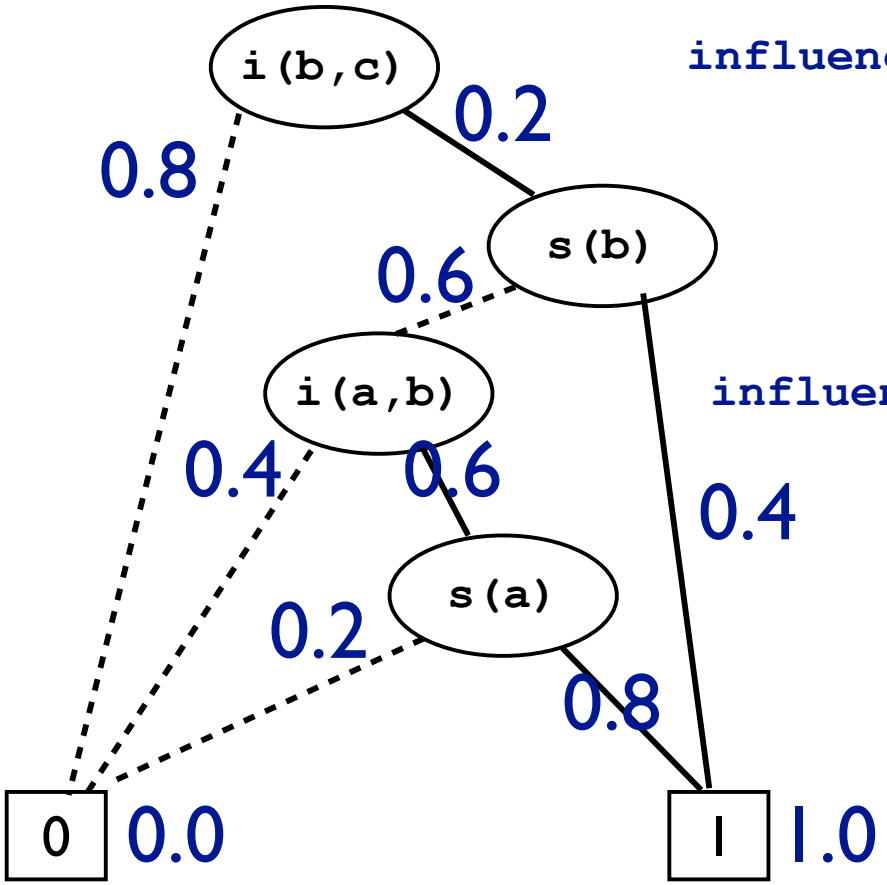
stress(ann)?

probability of

smokes(c)?

smokes(c) = i(b,c)&s(b) v
i(b,c)&i(a,b)&s(a)

Binary Decision Diagrams

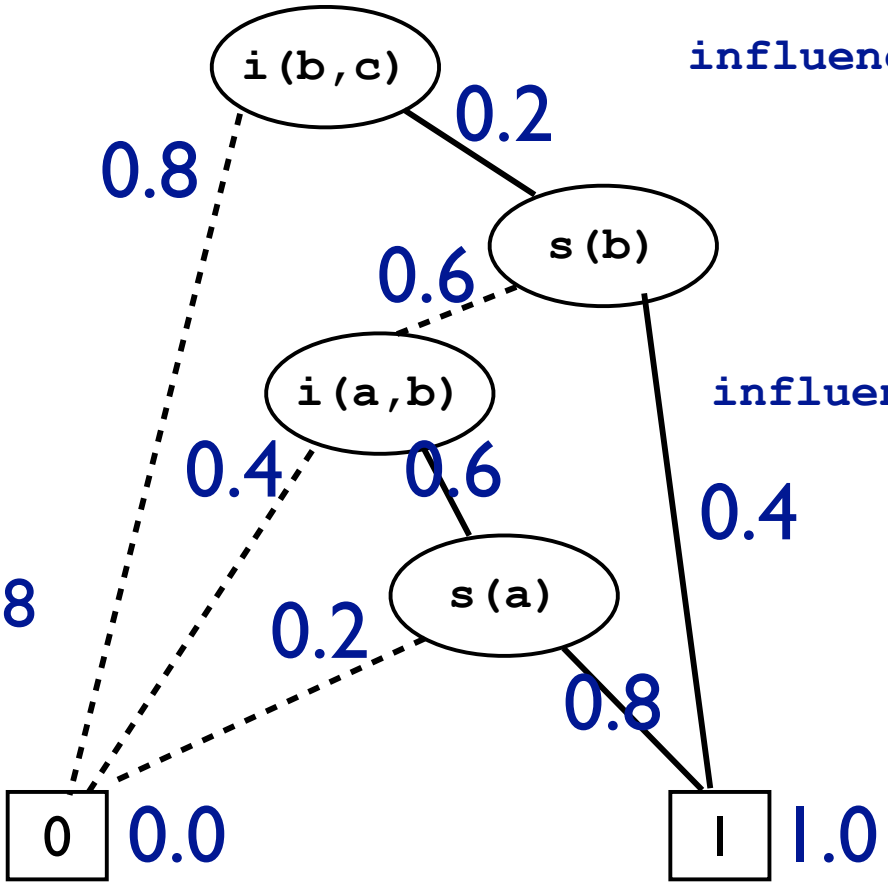


probability of

`smokes(c)`?

`smokes(c) = i(b,c)&s(b) v`
`i(b,c)&i(a,b)&s(a)`

Binary Decision Diagrams



influences(bob,carl)?

stress(bob)?

influences(ann,bob)?

stress(ann)?

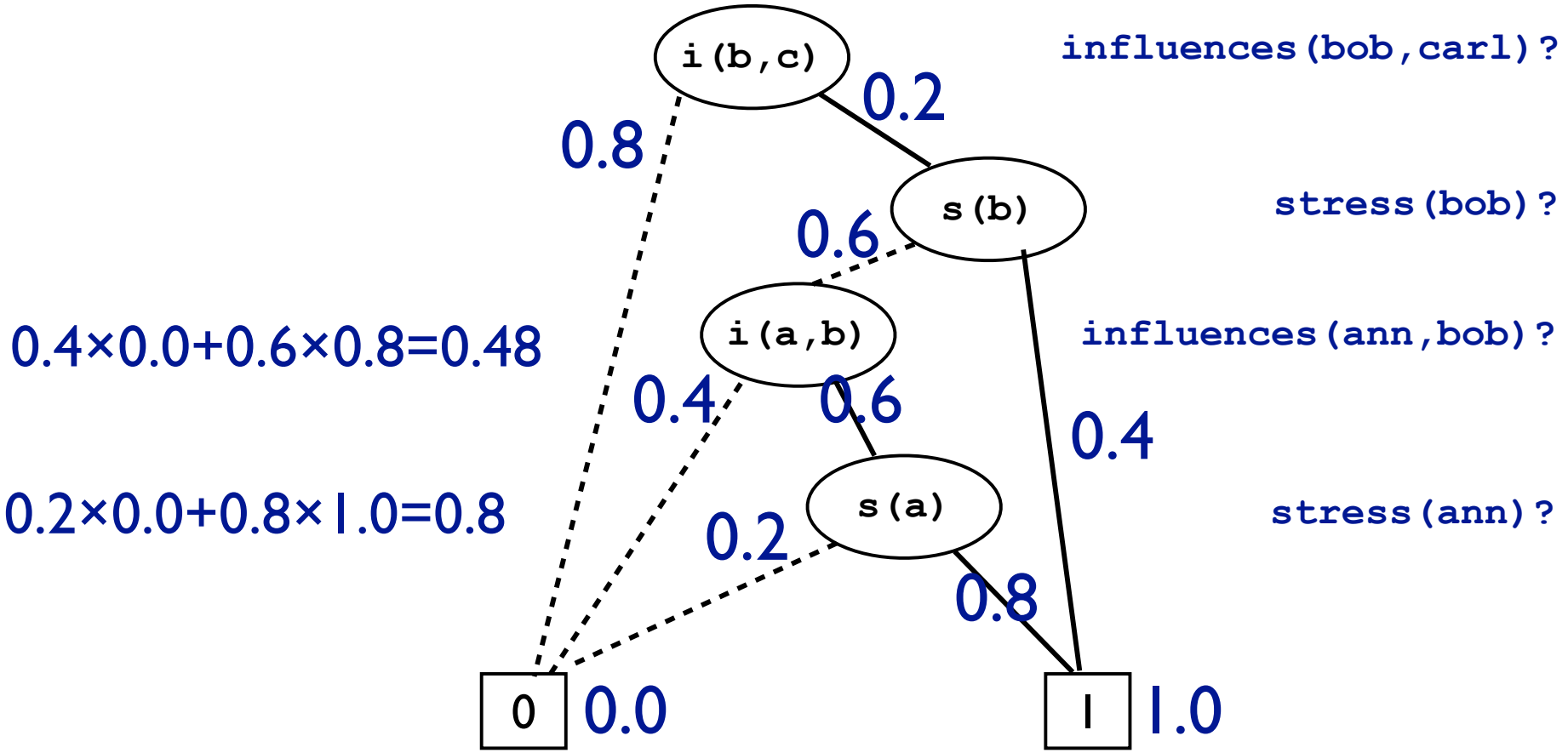
0.2×0.0+0.8×1.0=0.8

probability of

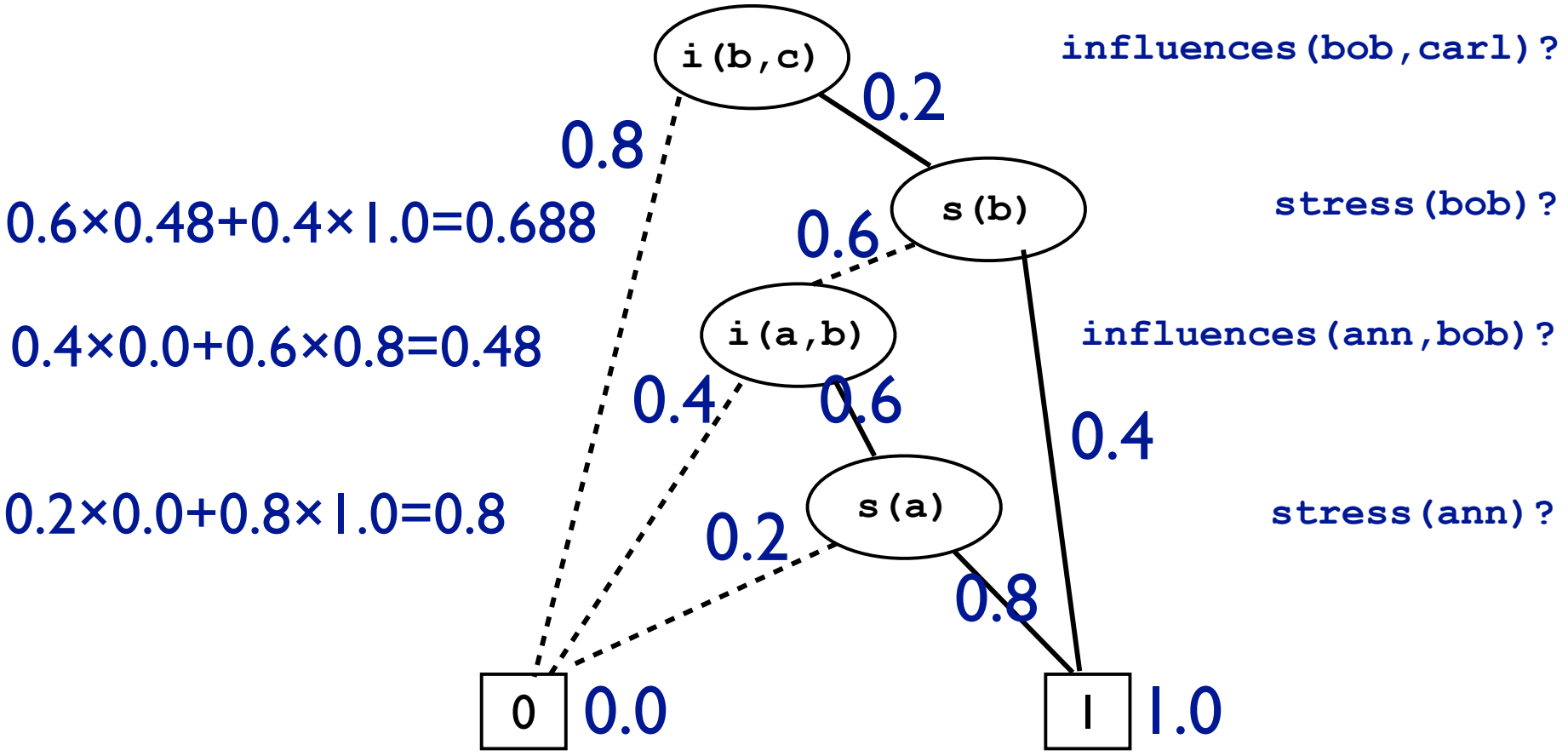
smokes(c)?

smokes(c) = i(b,c)&s(b) v
i(b,c)&i(a,b)&s(a)

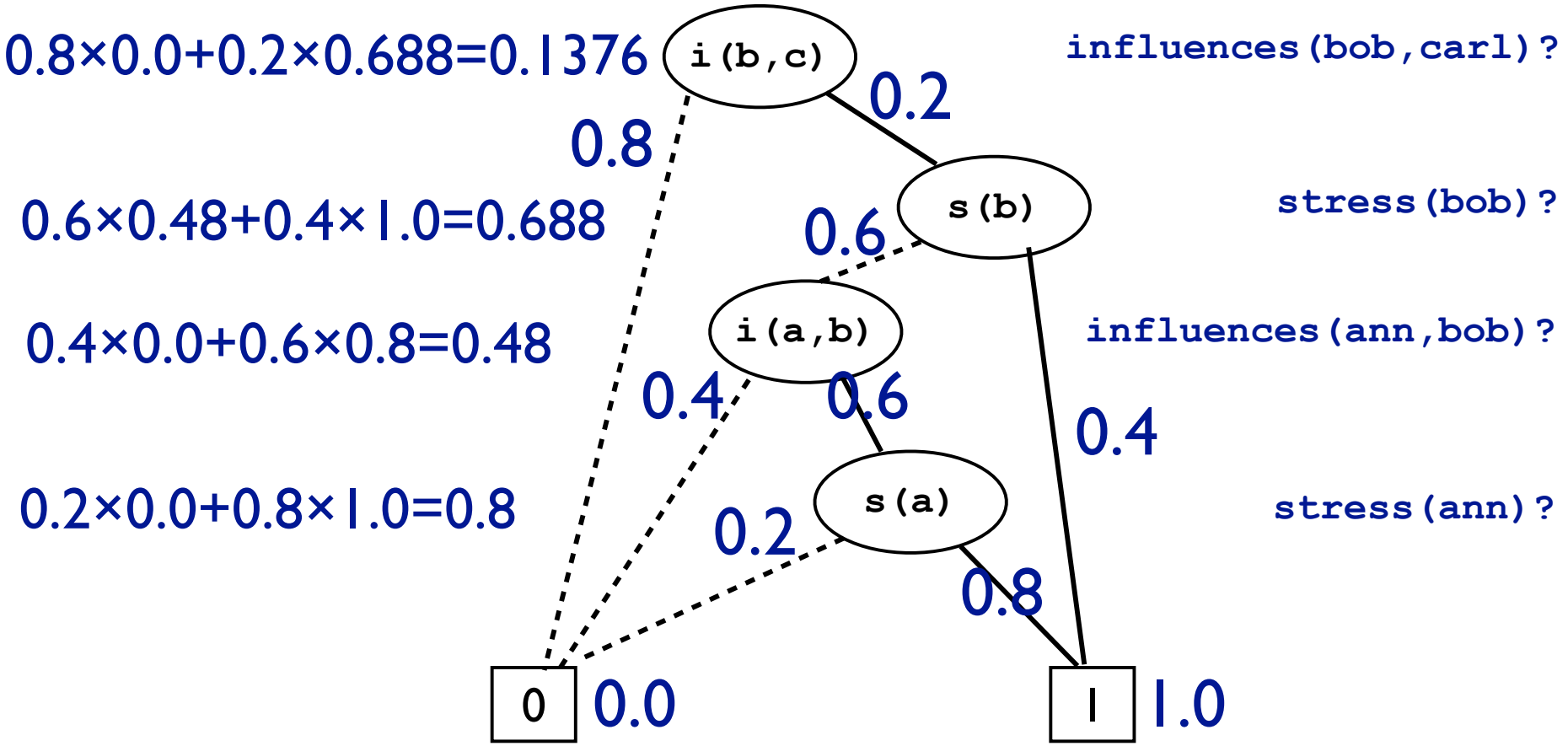
Binary Decision Diagrams



Binary Decision Diagrams

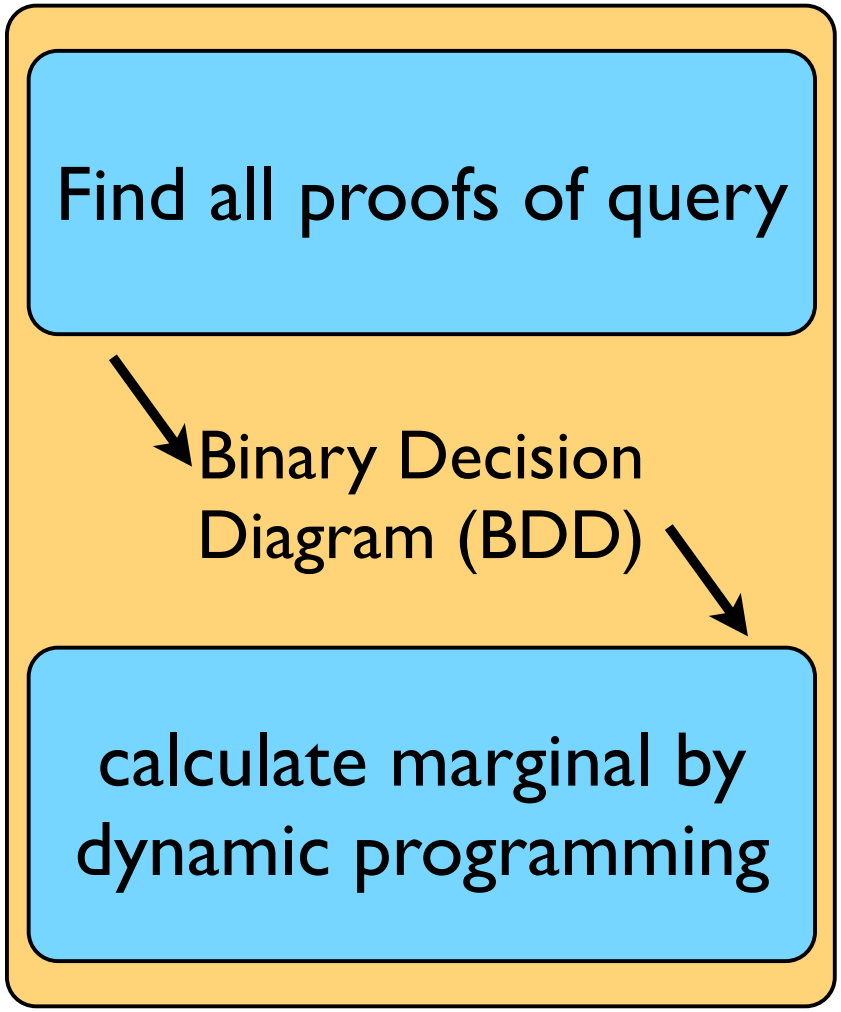


Binary Decision Diagrams



Initial Approach

(ProbLog1 & others)

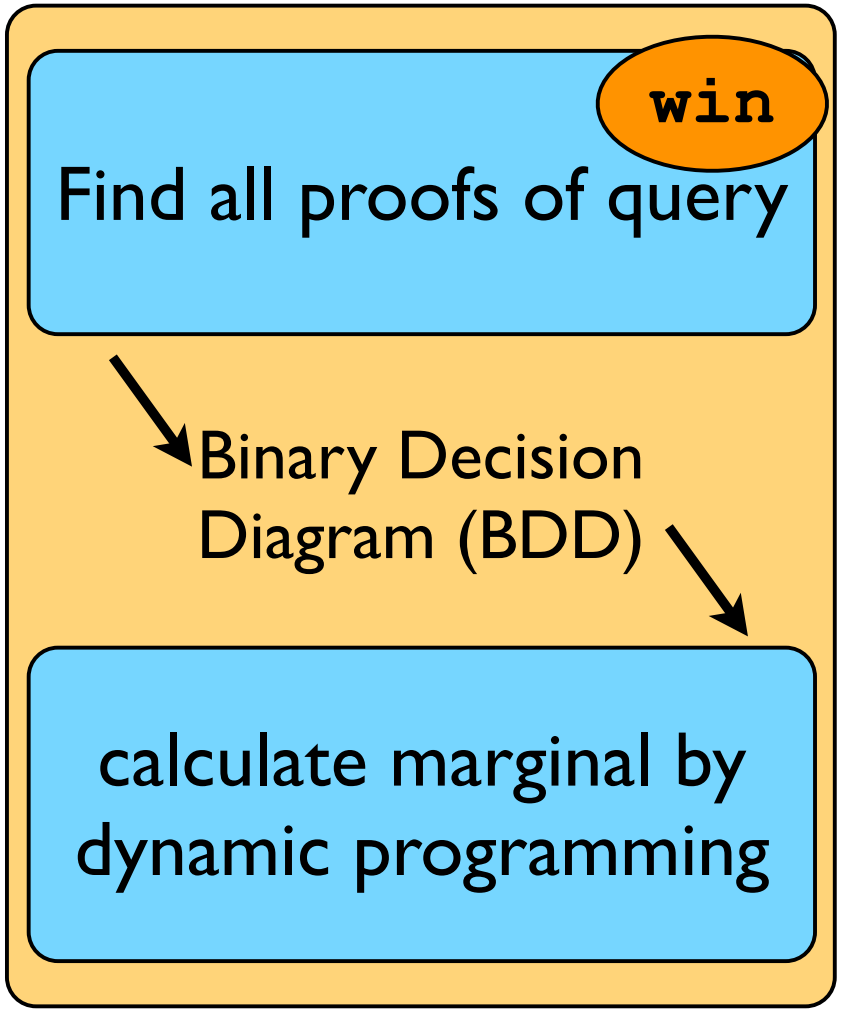


[De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

Initial Approach

(ProbLog1 & others)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),heads(3).
```

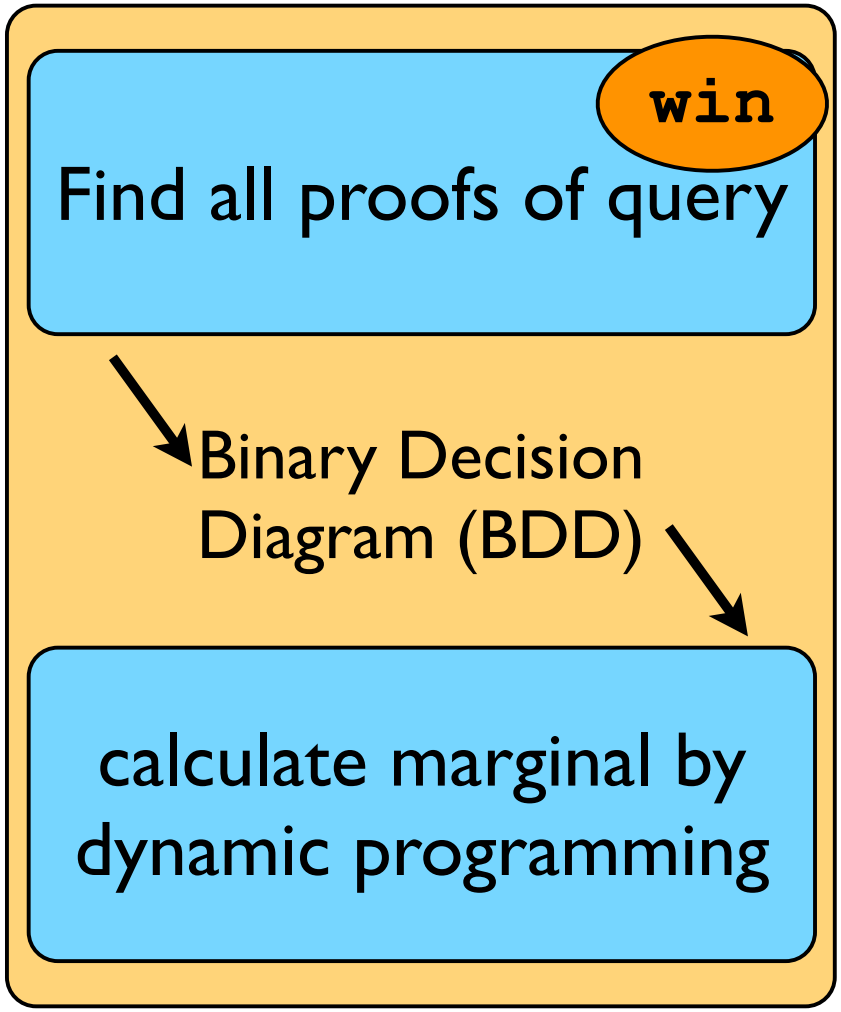


[De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

Initial Approach

(ProbLog1 & others)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),heads(3).
```



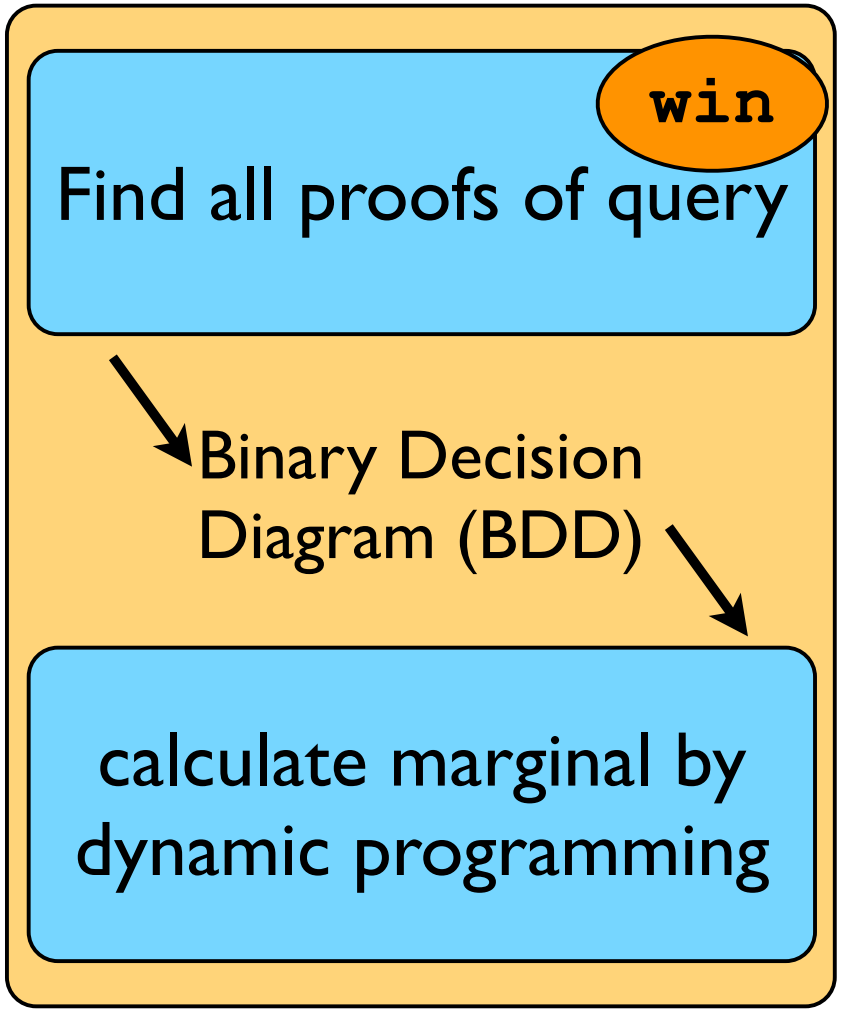

```
heads(1)
heads(2)  & heads(3)
```


[De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

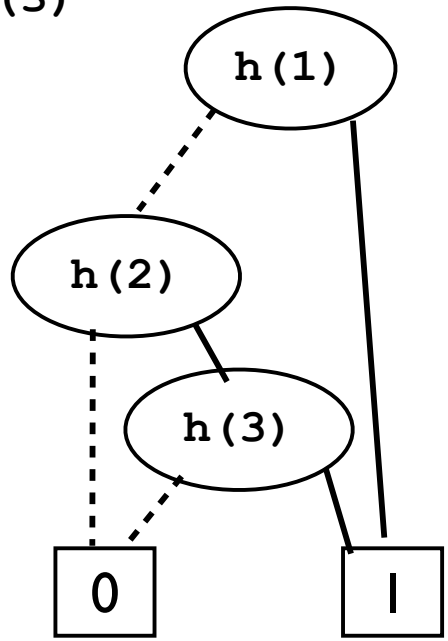
Initial Approach

(ProbLog1 & others)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),heads(3).
```




```
heads(1)
heads(2)  & heads(3)
```

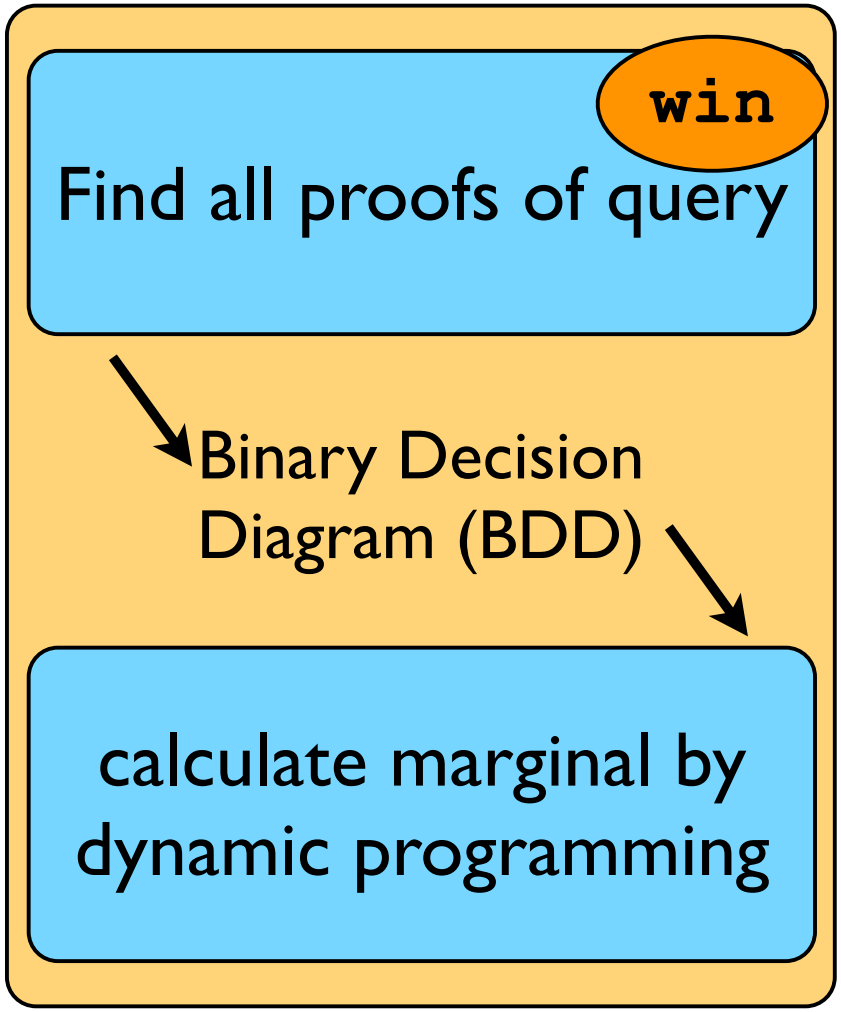


 [De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

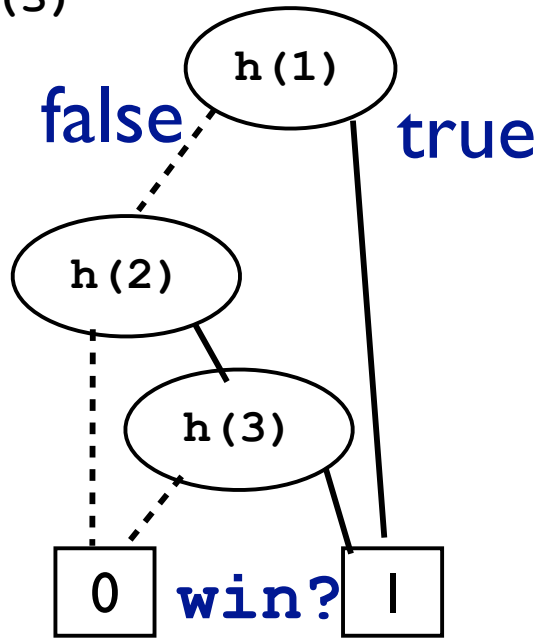
Initial Approach

(ProbLog1 & others)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),heads(3).
```




```
heads(1)
heads(2)  & heads(3)
```

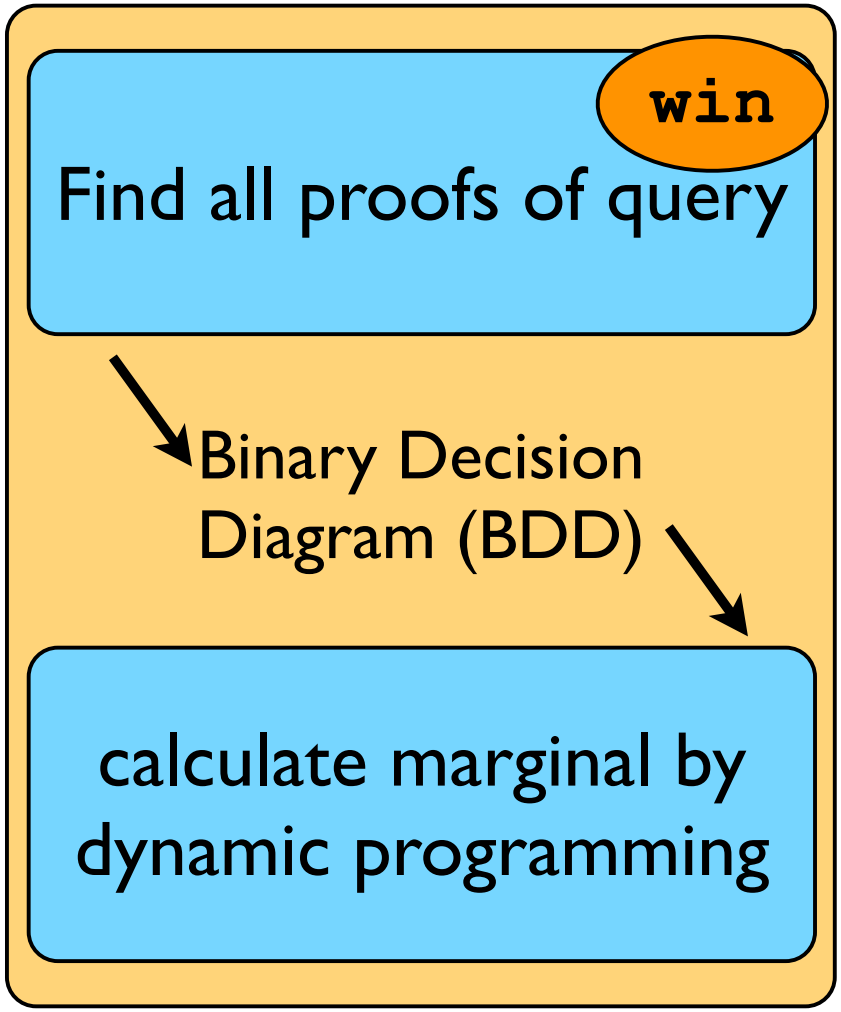


[De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

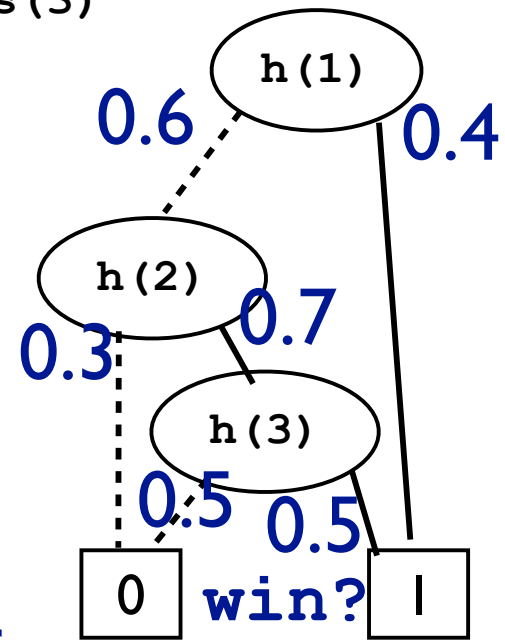
Initial Approach

(ProbLog1 & others)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),heads(3).
```



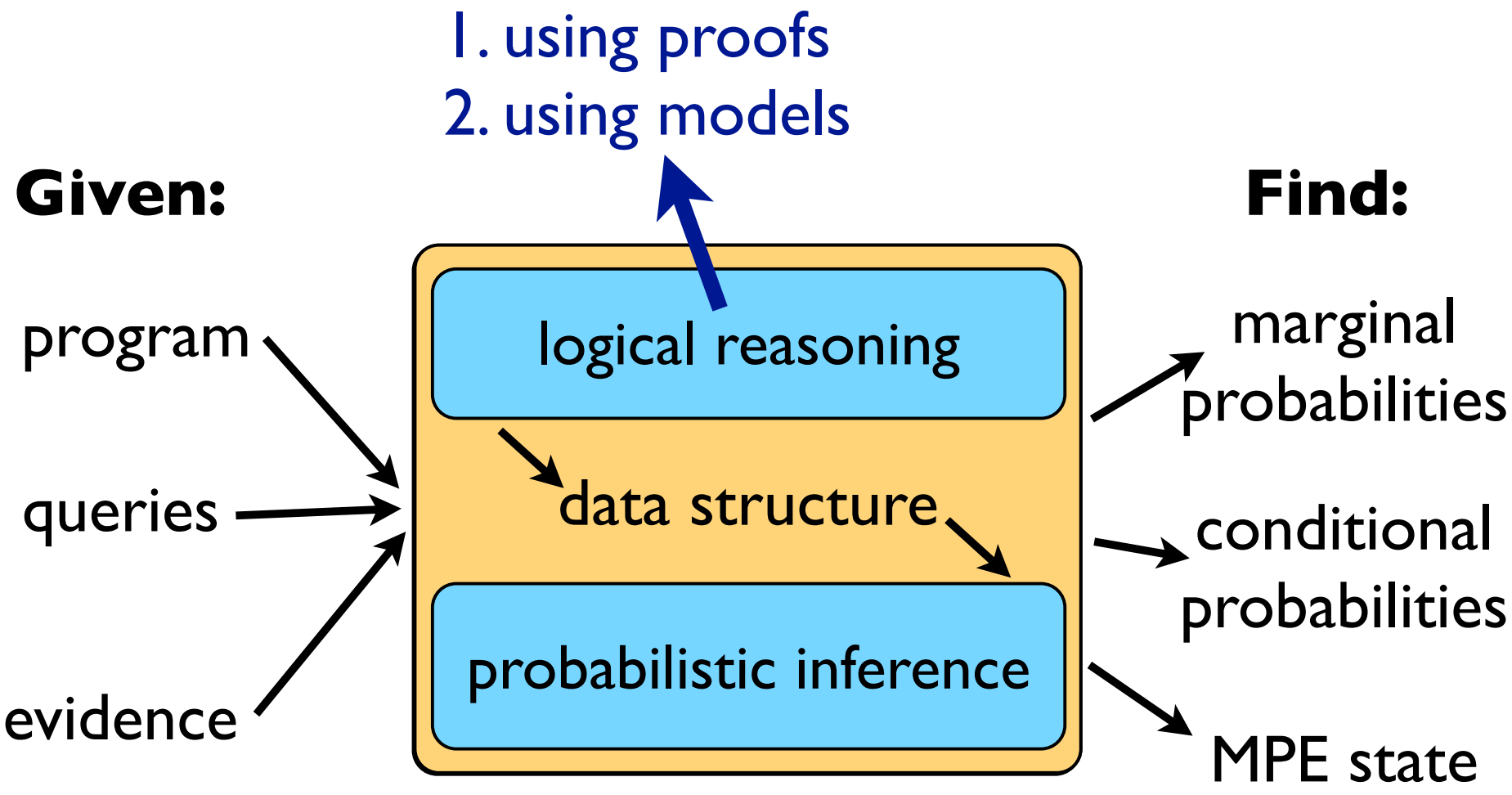

```
heads(1)
heads(2)  & heads(3)
```



P(**win**) = probability of reaching 1-leaf

[De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

Answering Questions



Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(ann).
```

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(ann).
smokes(bob).
```

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(ann).
smokes(bob).
smokes(carl).
```

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts
- Query true iff in model

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(ann).
smokes(bob).
smokes(carl).
```

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts
- Query true iff in model
- **ProbLog**: each possible world is a model, probability of query is sum over models where query is true

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(ann).
smokes(bob).
smokes(carl).
```

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts
- Query true iff in model

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(ann).
smokes(bob).
smokes(carl).
```

- **ProbLog**: each possible world is a model, probability of query is sum over models where query is true

→ weighted model counting

Weighted Model Counting

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I_V} w(l)$$

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I_V} w(l)$$

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I_V} w(l)$$

interpretations (truth value assignments) of propositional variables

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I_V} w(l)$$

interpretations (truth value assignments) of propositional variables

weight of literal

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

given by ProbLog program & query

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I_V} w(l)$$

interpretations (truth value assignments) of propositional variables

weight of literal

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

given by ProbLog program & query

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I_V} w(l)$$

interpretations (truth value assignments) of propositional variables

possible worlds

weight of literal

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

given by ProbLog program & query

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I_V} w(l)$$

interpretations (truth value assignments) of propositional variables

possible worlds

weight of literal

for p::f,
w(f) = p
w(not f) = 1−p

Weighte

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

propositional formula in conjunctive normal form (CNF)

given by ProbLog program & query

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I_V} w(l)$$

interpretations (truth value assignments) of propositional variables

possible worlds

weight of literal

for p::f,
w(f) = p
w(not f) = 1−p

ProbLog →CNF

```
?- smokes(carl).
```

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```

ProbLog →CNF

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```

```
?- smokes(carl).
```

- Find relevant ground rules by backward reasoning

ProbLog → CNF

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```

```
?- smokes(carl).
```

- Find relevant ground rules by backward reasoning

```
smokes(carl) :- influences(bob,carl),smokes(bob).
smokes(bob) :- stress(bob).
smokes(bob) :- influences(ann,bob),smokes(ann).
smokes(ann) :- stress(ann).
```

ProbLog →CNF

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```

```
?- smokes(carl).
```

- Find relevant ground rules by backward reasoning

```
smokes(carl) :- influences(bob,carl),smokes(bob).
smokes(bob) :- stress(bob).
smokes(bob) :- influences(ann,bob),smokes(ann).
smokes(ann) :- stress(ann).
```

- Convert to propositional logic formula

ProbLog →CNF

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```

```
?- smokes(carl).
```

- Find relevant ground rules by backward reasoning

```
smokes(carl) :- influences(bob,carl),smokes(bob).
smokes(bob) :- stress(bob).
smokes(bob) :- influences(ann,bob),smokes(ann).
smokes(ann) :- stress(ann).
```

- Convert to propositional logic formula

may require loop-breaking

$$sm(c) \leftrightarrow (i(b,c) \wedge sm(b))$$
$$\wedge\ sm(b) \leftrightarrow (st(b) \vee (i(a,b) \wedge sm(a)))$$
$$\wedge\ sm(a) \leftrightarrow st(a)$$

ProbLog →CNF

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```

```
?- smokes(carl).
```

- Find relevant ground rules by backward reasoning

```
smokes(carl) :- influences(bob,carl),smokes(bob).
smokes(bob) :- stress(bob).
smokes(bob) :- influences(ann,bob),smokes(ann).
smokes(ann) :- stress(ann).
```

- Convert to propositional logic formula

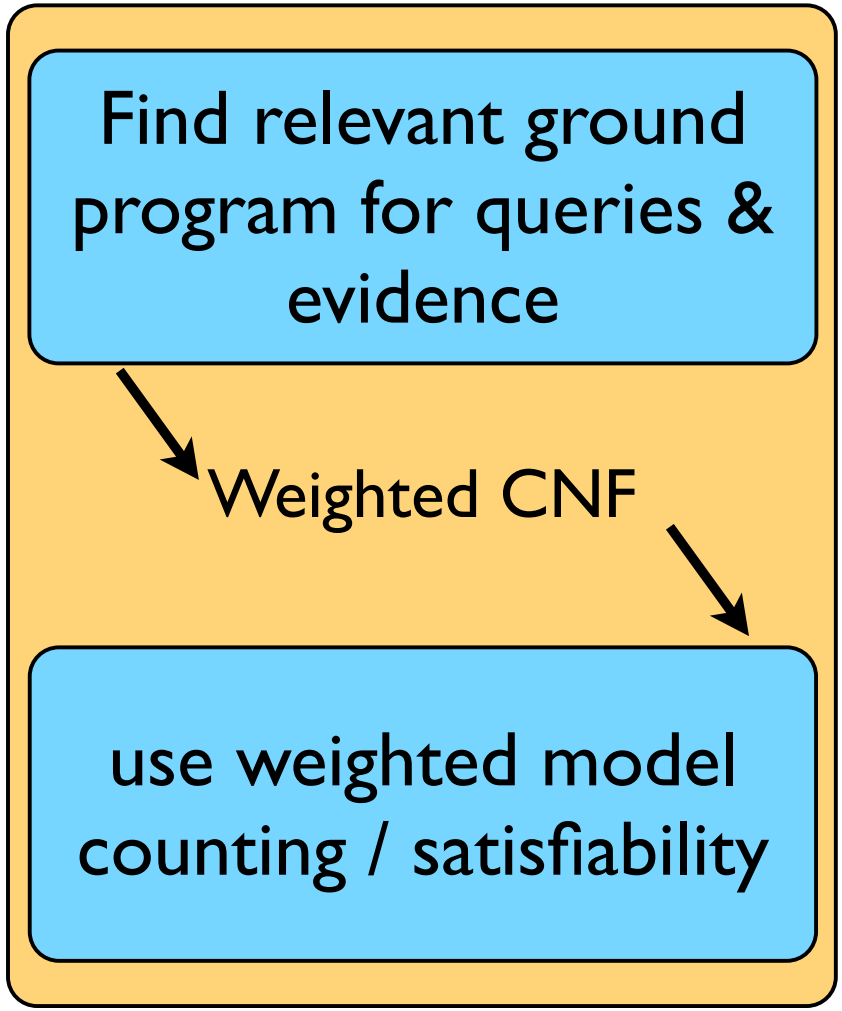
may require loop-breaking

$$sm(c) \leftrightarrow (i(b,c) \wedge sm(b))$$
$$\wedge\ sm(b) \leftrightarrow (st(b) \vee (i(a,b) \wedge sm(a)))$$
$$\wedge\ sm(a) \leftrightarrow st(a)$$

- Rewrite in CNF (as usual)

Current Approach

(ProbLog2)

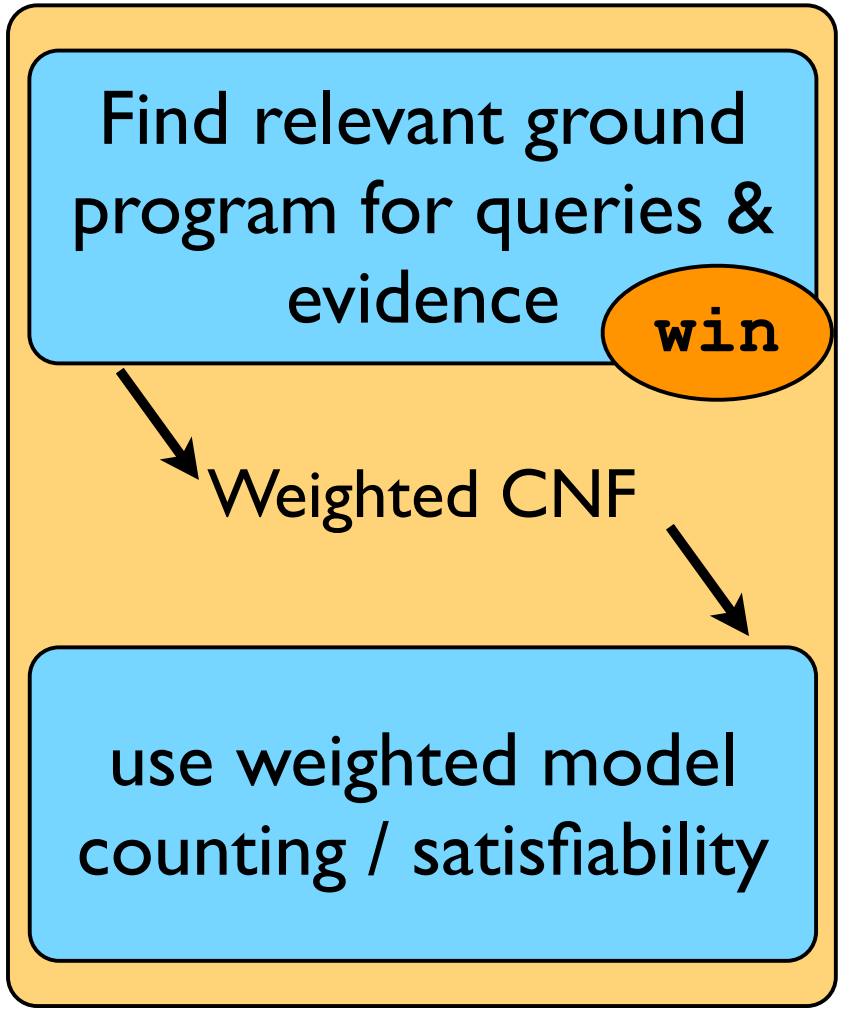


[Fierens et al, TPLP 14]

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```

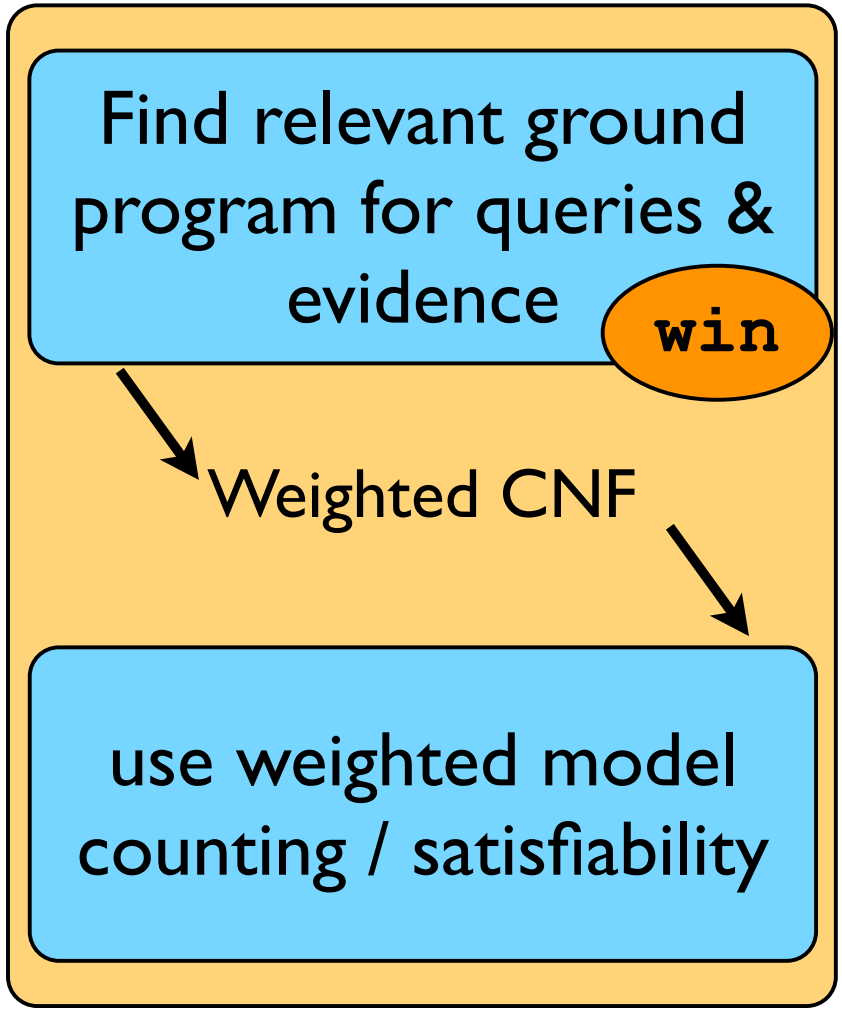


[Fierens et al, TPLP 14]

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```



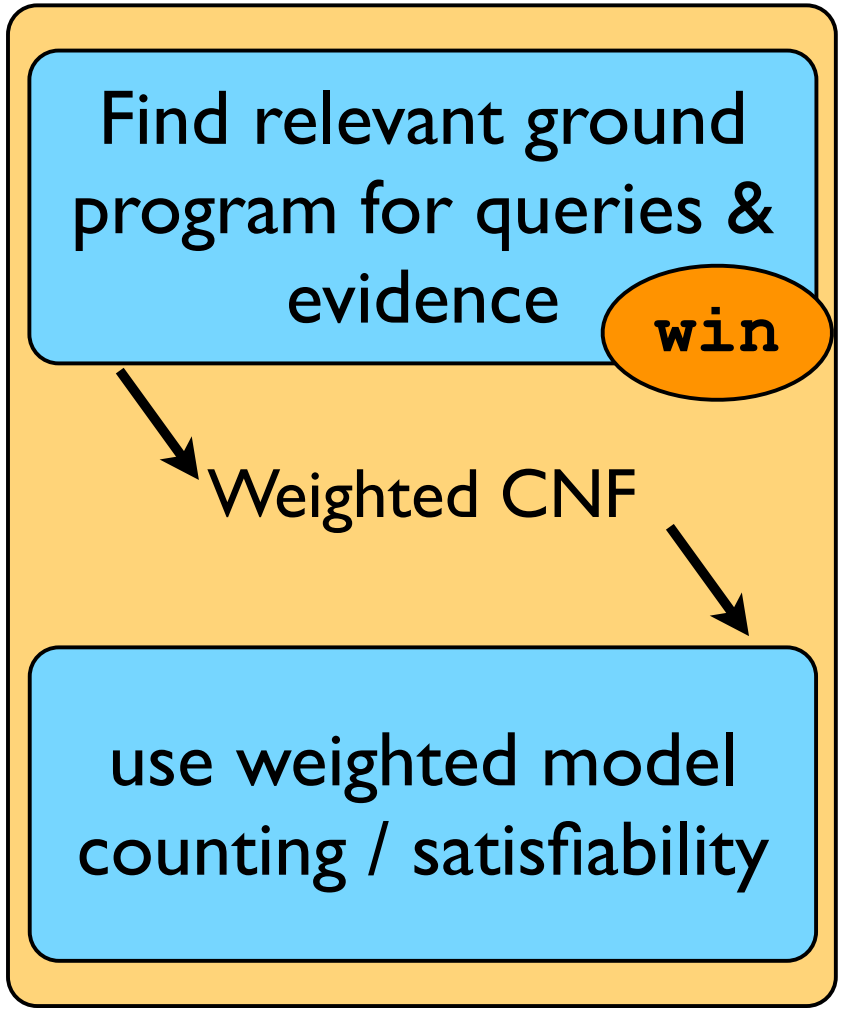

```
win :- heads(1).
win :- heads(2), heads(3).
```


[Fierens et al, TPLP 14]

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```




```
win :- heads(1).
win :- heads(2), heads(3).
```

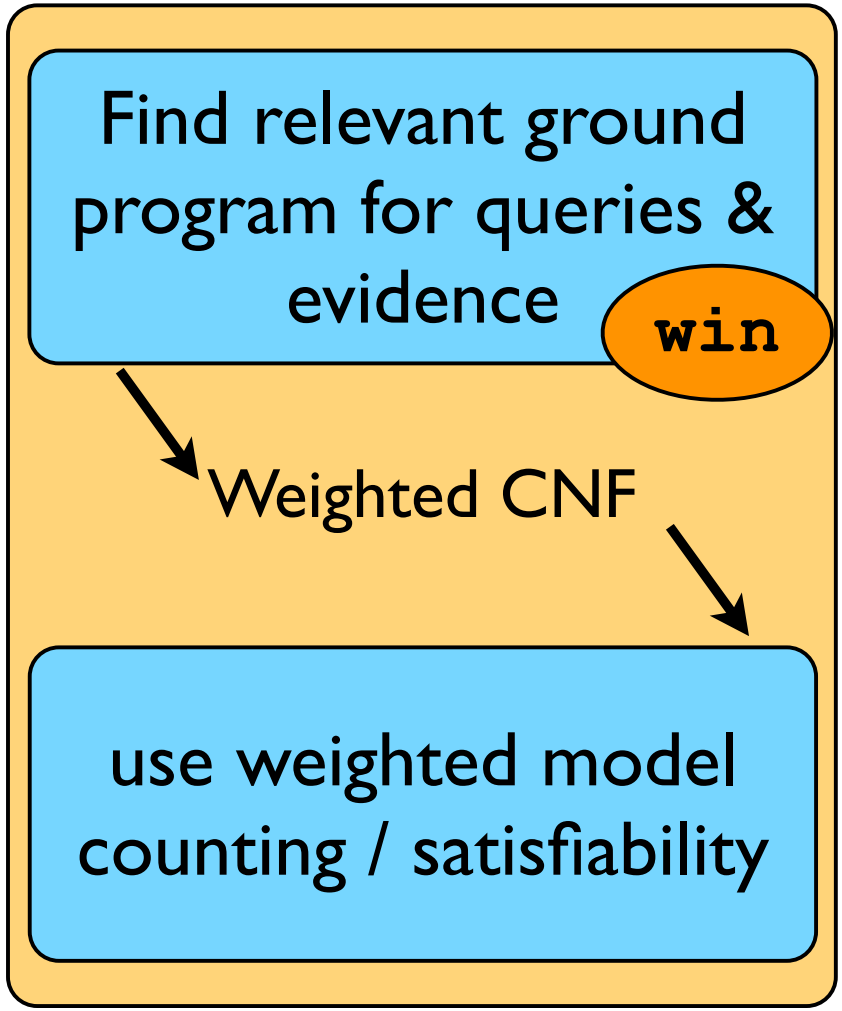
$$\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$$

[Fierens et al, TPLP 14]

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```




```
win :- heads(1).
win :- heads(2), heads(3).
```

$$\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$$

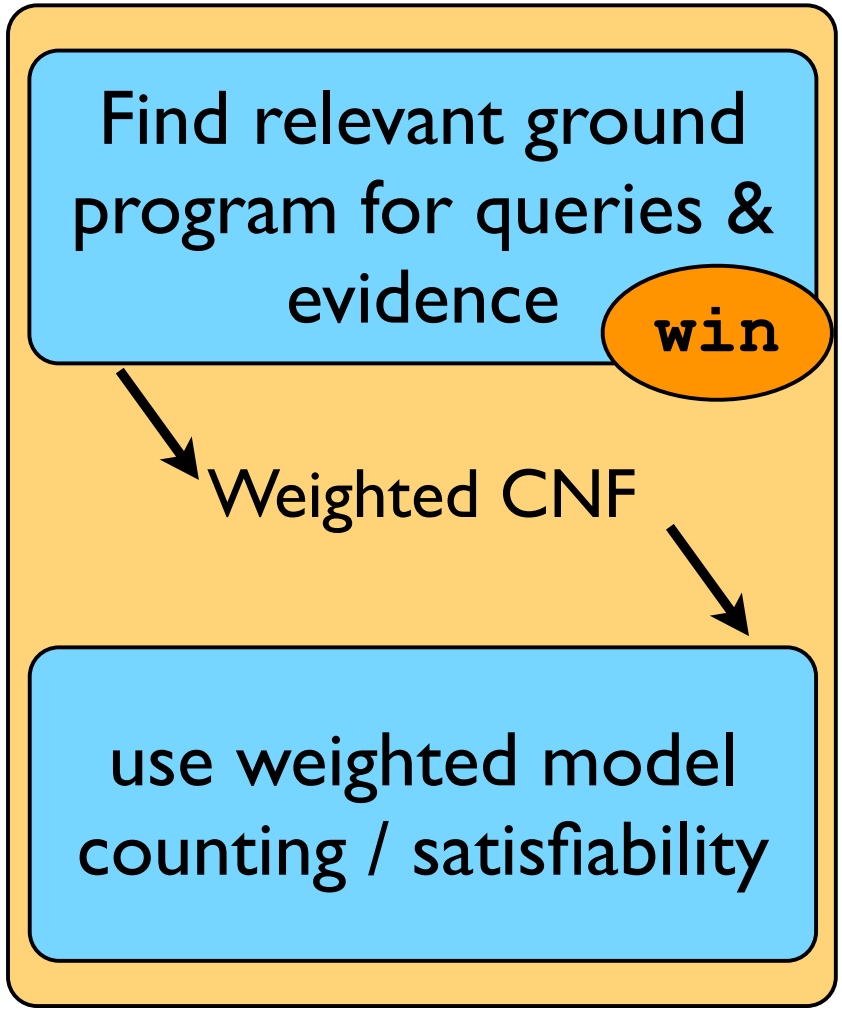
$$(\neg\text{win} \vee h(1) \vee h(2)) \wedge (\neg\text{win} \vee h(1) \vee h(3)) \wedge (\text{win} \vee \neg h(1)) \wedge (\text{win} \vee \neg h(2) \vee \neg h(3))$$

[Fierens et al, TPLP 14]

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```




```
win :- heads(1).
win :- heads(2), heads(3).
```

$$\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$$

$$(\neg\text{win} \vee h(1) \vee h(2)) \wedge (\neg\text{win} \vee h(1) \vee h(3)) \wedge (\text{win} \vee \neg h(1)) \wedge (\text{win} \vee \neg h(2) \vee \neg h(3))$$

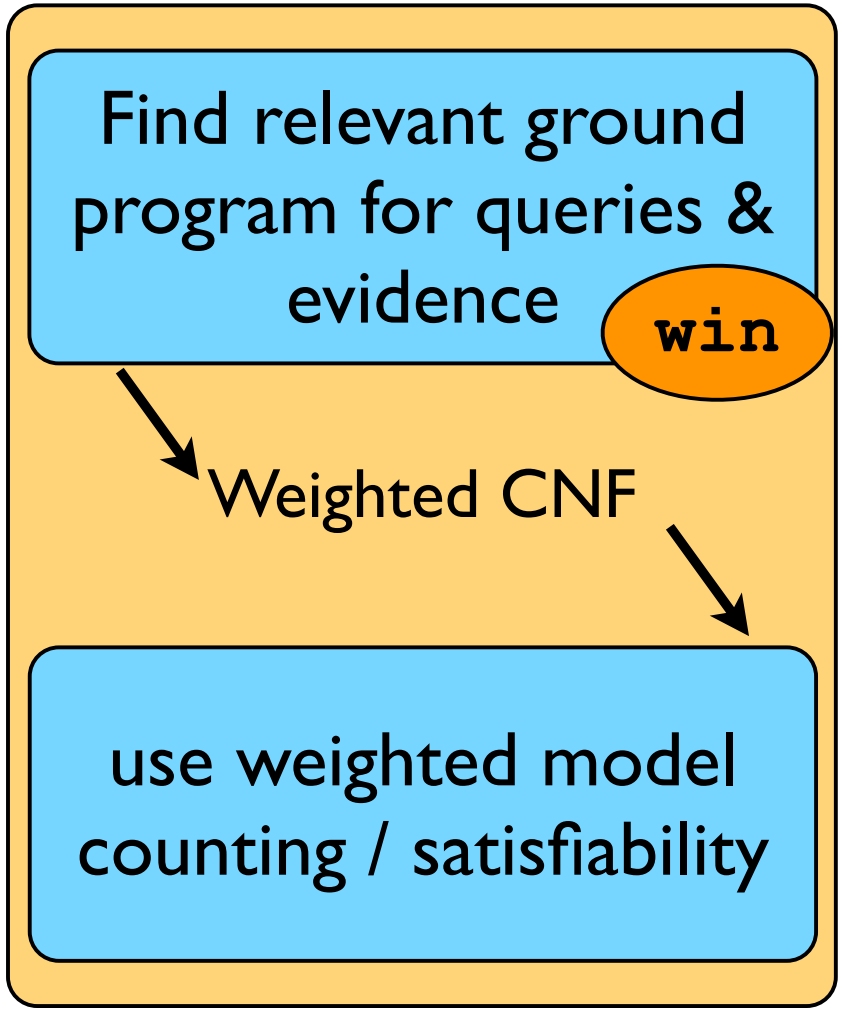
$h(1) \rightarrow 0.4$	$h(2) \rightarrow 0.7$	$h(3) \rightarrow 0.5$
$\neg h(1) \rightarrow 0.6$	$\neg h(2) \rightarrow 0.3$	$\neg h(3) \rightarrow 0.5$

[Fierens et al, TPLP 14]

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```




```
win :- heads(1).
win :- heads(2), heads(3).
```

win ↔ h(1) ∨ (h(2) ∧ h(3))

(¬win ∨ h(1) ∨ h(2))
∧ (¬win ∨ h(1) ∨ h(3))
∧ (win ∨ ¬h(1))
∧ (win ∨ ¬h(2) ∨ ¬h(3))

use standard tool

h(1) → 0.4	h(2) → 0.7	h(3) → 0.5
¬h(1) → 0.6	¬h(2) → 0.3	¬h(3) → 0.5

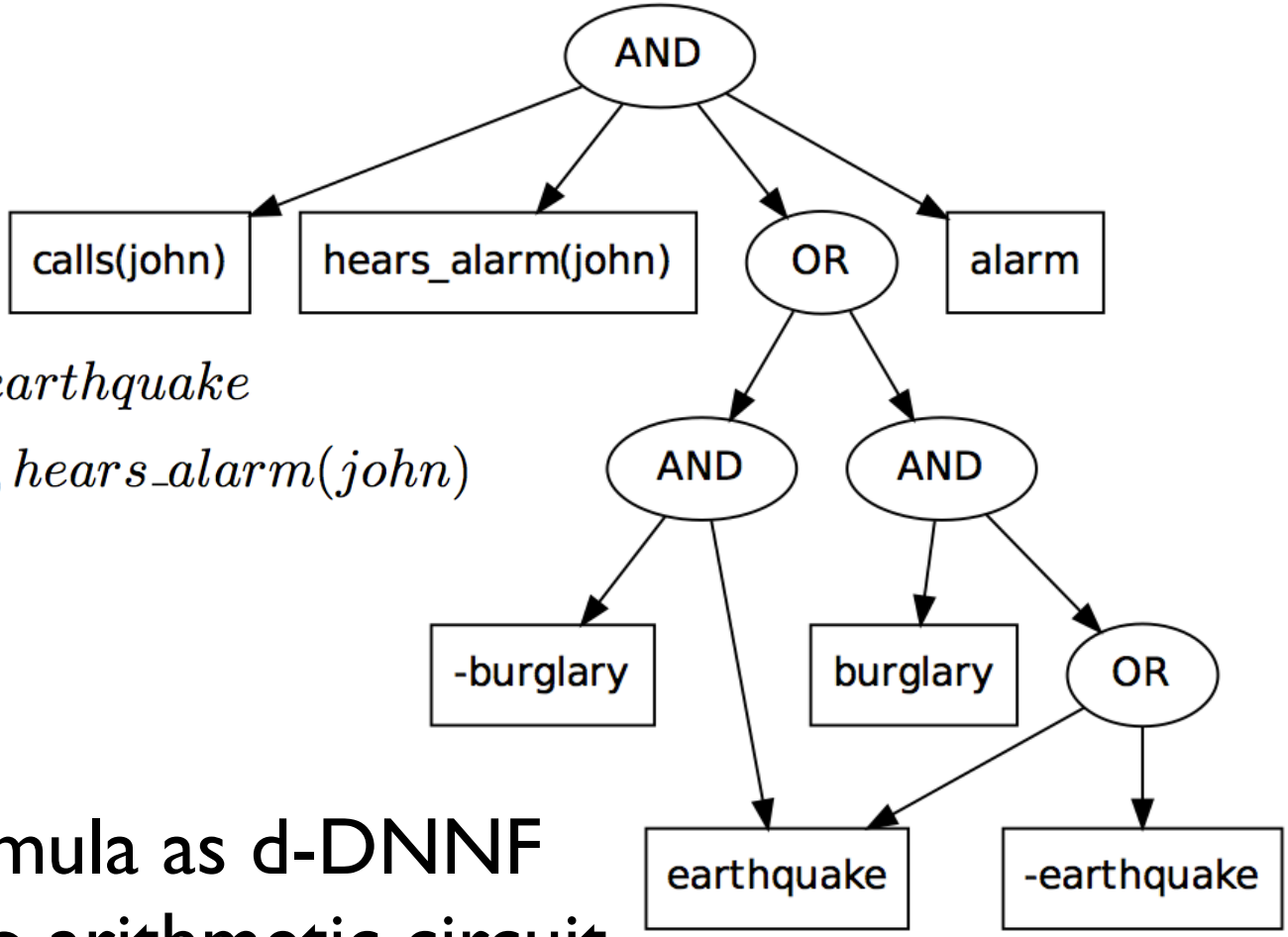
[Fierens et al, TPLP 14]

WMC using d-DNNFs

$$alarm \leftrightarrow burglary \vee earthquake$$

$$calls(john) \leftrightarrow alarm, hears\_alarm(john)$$

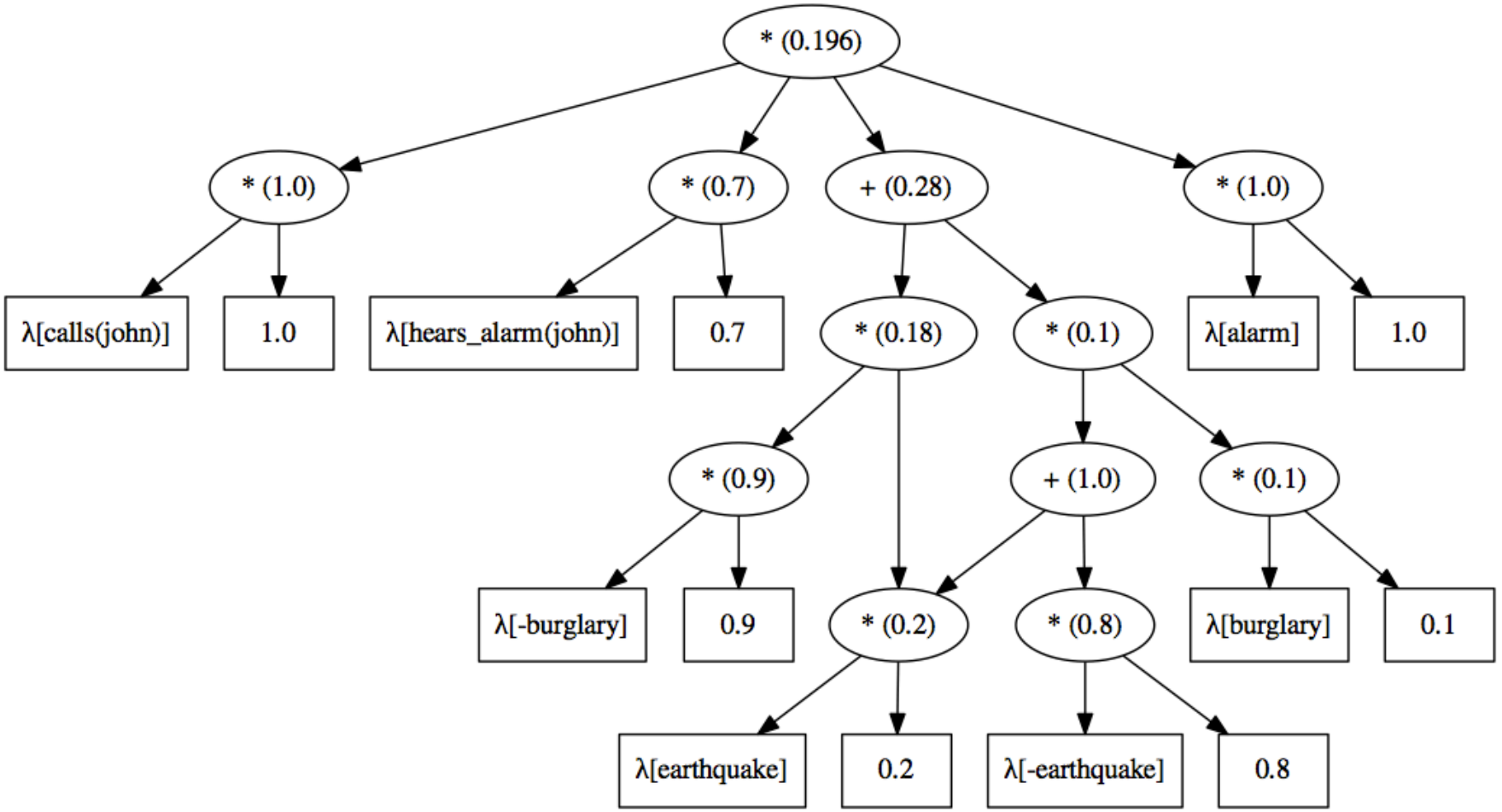
$$calls(john)$$



1. represent formula as d-DNNF
2. transform into arithmetic circuit
3. evaluate bottom-up

[Figure: Fierens et al, TPLP 14]

WMC using d-DNNFs



3. evaluate bottom-up

[Figure: Fierens et al, TPLP 14]

ProbLog Inference

- reduction to propositional formula
- addresses disjoint-sum-problem
- **but**: not all probabilistic logic programs face this problem! e.g., weather
- more generally: mutually exclusive proofs as assumed in PRISM

PRISM: compute probability by dynamic programming

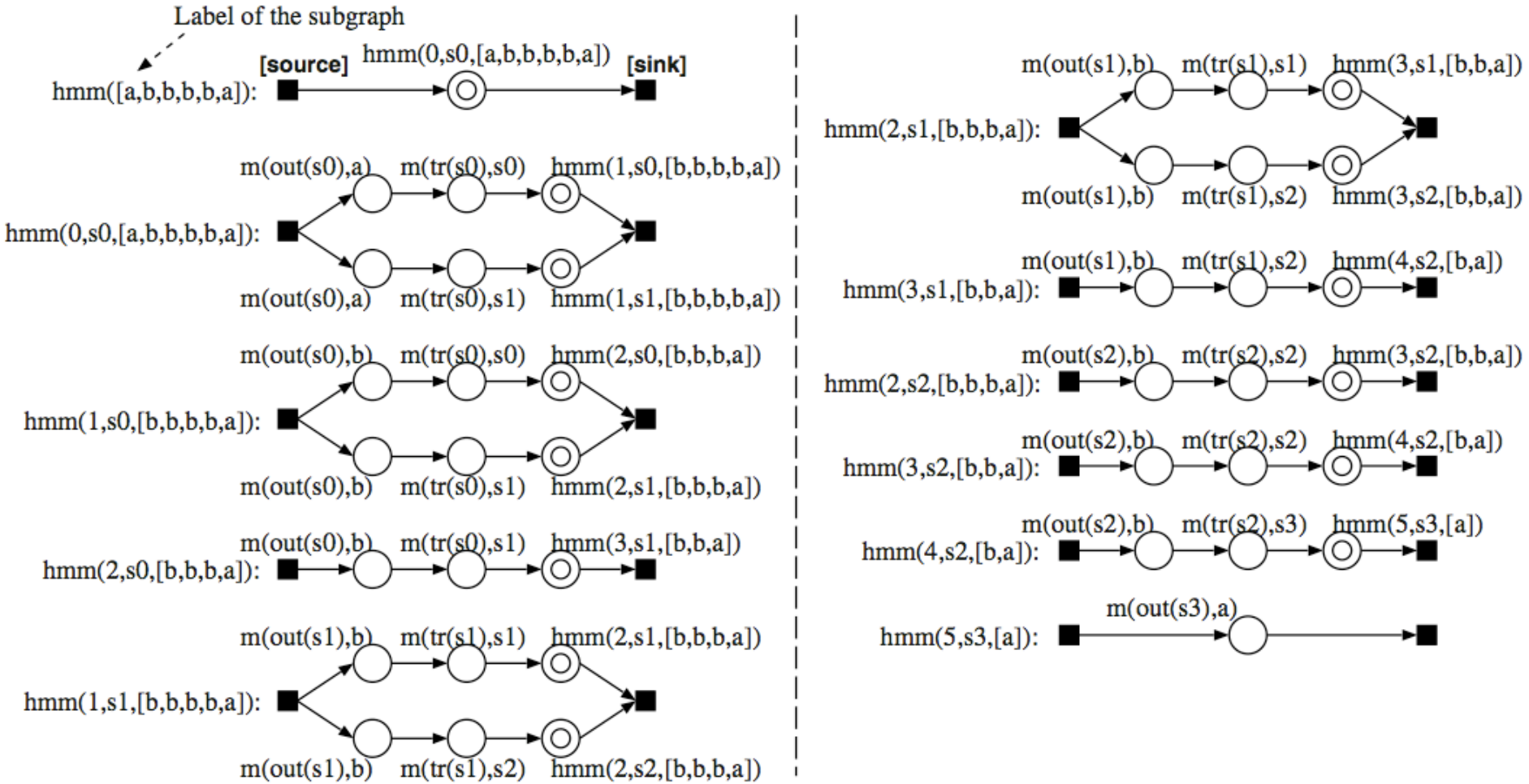


Fig. 5. Explanation graph

[Figures: Sato and Kameya 08]

PRISM: compute probability by dynamic programming

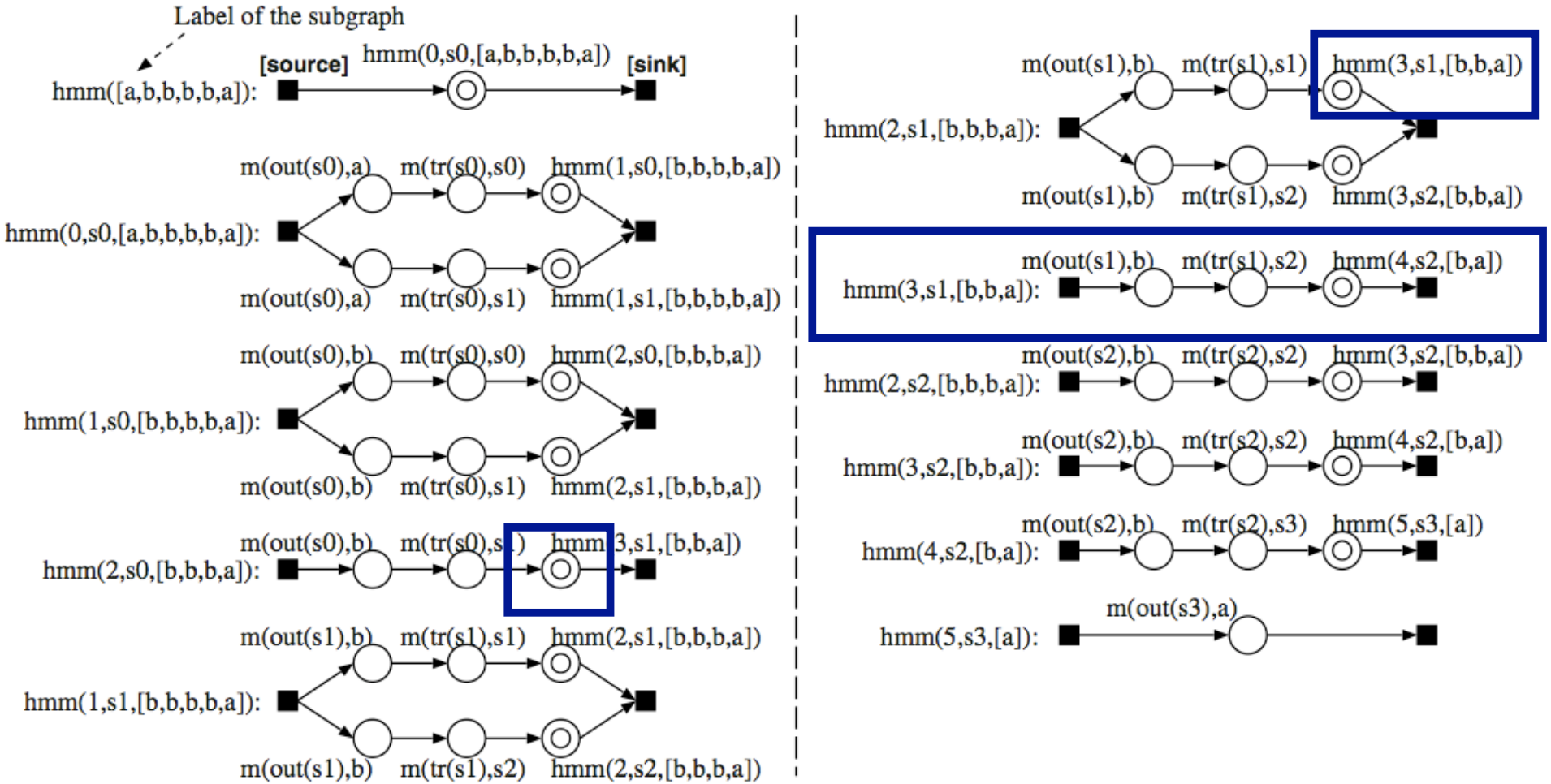


Fig. 5. Explanation graph

[Figures: Sato and Kameya 08]

Dyna (Eisner et al.)

word(john,0,1), word(loves,1,2), word(Mary,2,3)
0.003:: np -> Mary as rewrite(np,Mary)=0.003
0.5::vp -> verb, np as rewrite(vp,verb,np)=0.5

- CKY Algorithm in Dyna

```
constit(X,I,J) += rewrite(X,W) * word(W,I,J).
constit(X,I,K) += rewrite(X,Y,Z) * constit(Y,I,J) * constit(Z,J,K).
goal += constit(S,0,N) * end(N).
```

- 

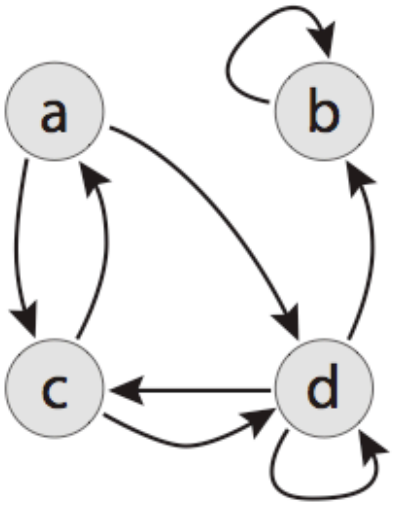
Origins in Natural Language Processing Context
Based on Dynamic Programming
Useful for SRL/NLP/Programming

Dyna

- Weighted Logic Programs (but not Prolog)
- Inspired by NLP
- Arbitrary semiring weights
- Forward reasoning

```
reachable(Q) :- initial(Q).
reachable(Q) :- reachable(P), edge(P, Q).
```

Fig. 1. A simple bottom-up logic program for graph reachability



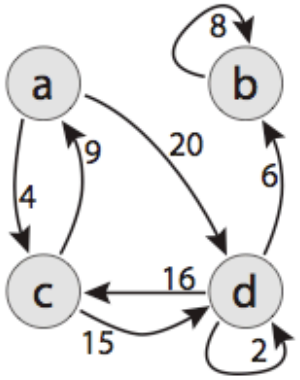

```
initial(a) = T   edge(c, d) = T
 edge(a, c) = T   edge(d, b) = T
 edge(a, d) = T   edge(d, c) = T
 edge(b, b) = T   edge(d, d) = T
 edge(c, a) = T
```

Fig. 2. A directed graph and the corresponding initial database

[Figures: Cohen et al, ICLP 08]

$$\texttt{reachable(Q)} \oplus\!= \texttt{initial(Q)}.$$

$$\texttt{reachable(Q)} \oplus\!= \texttt{reachable(P)} \otimes \texttt{edge(P, Q)}.$$



$$\begin{array}{ll} \texttt{initial(a)} = 0 & \texttt{edge(c, d)} = 15 \\ \texttt{edge(a, c)} = 4 & \texttt{edge(d, b)} = 6 \\ \texttt{edge(a, d)} = 20 & \texttt{edge(d, c)} = 16 \\ \texttt{edge(b, b)} = 8 & \texttt{edge(d, d)} = 2 \\ \texttt{edge(c, a)} = 9 & \end{array}$$

Fig. 3. A cost graph and the corresponding initial database

[Figures: Cohen et al, ICLP 08]

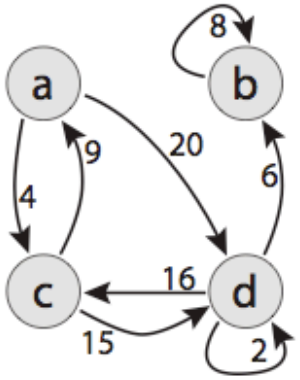
$$\texttt{reachable(Q)} \oplus\!\!= \texttt{initial(Q)}.$$

$$\texttt{reachable(Q)} \oplus\!\!= \texttt{reachable(P)} \otimes \texttt{edge(P, Q)}.$$

$\oplus \rightarrow \min$

$\otimes \rightarrow +$

shortest path



$\texttt{initial(a)} = 0$
$\texttt{edge(a, c)} = 4$
$\texttt{edge(a, d)} = 20$
$\texttt{edge(b, b)} = 8$
$\texttt{edge(c, a)} = 9$
$\texttt{edge(c, d)} = 15$
$\texttt{edge(d, b)} = 6$
$\texttt{edge(d, c)} = 16$
$\texttt{edge(d, d)} = 2$

Fig. 3. A cost graph and the corresponding initial database

[Figures: Cohen et al, ICLP 08]

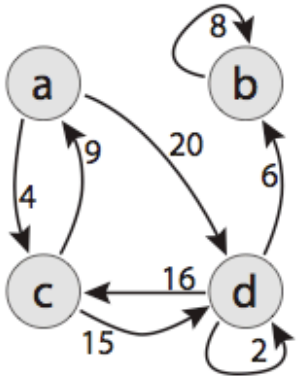
$$\texttt{reachable(Q)} \oplus\!= \texttt{initial(Q)}.$$

$$\texttt{reachable(Q)} \oplus\!= \texttt{reachable(P)} \otimes \texttt{edge(P, Q)}.$$

$\oplus \rightarrow \min$

$\otimes \rightarrow +$

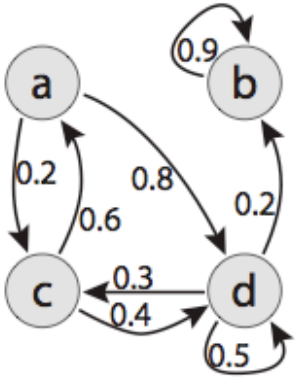
shortest path



$\texttt{initial(a)} = 0$ $\quad \texttt{edge(a, c)} = 4$ $\quad \texttt{edge(a, d)} = 20$ $\quad \texttt{edge(b, b)} = 8$ $\quad \texttt{edge(c, a)} = 9$

$\texttt{edge(c, d)} = 15$ $\quad \texttt{edge(d, b)} = 6$ $\quad \texttt{edge(d, c)} = 16$ $\quad \texttt{edge(d, d)} = 2$

Fig. 3. A cost graph and the corresponding initial database



$\texttt{initial(a)} = 1$ $\quad \texttt{edge(a, c)} = 0.2$ $\quad \texttt{edge(a, d)} = 0.8$ $\quad \texttt{edge(b, b)} = 0.9$ $\quad \texttt{edge(c, a)} = 0.6$

$\texttt{edge(c, d)} = 0.4$ $\quad \texttt{edge(d, b)} = 0.2$ $\quad \texttt{edge(d, c)} = 0.3$ $\quad \texttt{edge(d, d)} = 0.5$

Fig. 4. A probabilistic graph and the corresponding initial database. With stopping probabilities made explicit, this would encode a Markov model.

[Figures: Cohen et al, ICLP 08]

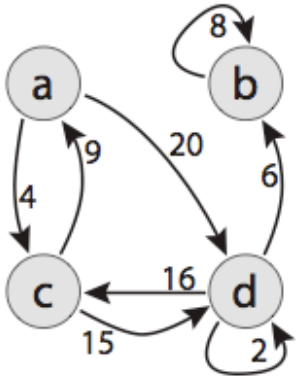
$$\texttt{reachable(Q)} \oplus\!= \texttt{initial(Q)}.$$

$$\texttt{reachable(Q)} \oplus\!= \texttt{reachable(P)} \otimes \texttt{edge(P, Q)}.$$

$\oplus \rightarrow \min$

$\otimes \rightarrow +$

shortest path



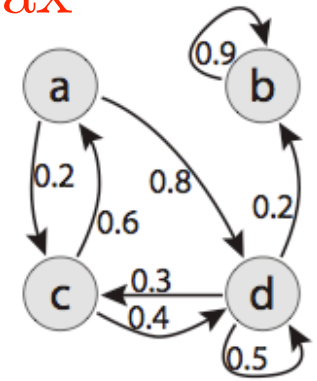
$$\texttt{initial(a)} = 0 \quad \texttt{edge(c, d)} = 15$$
$$\texttt{edge(a, c)} = 4 \quad \texttt{edge(d, b)} = 6$$
$$\texttt{edge(a, d)} = 20 \quad \texttt{edge(d, c)} = 16$$
$$\texttt{edge(b, b)} = 8 \quad \texttt{edge(d, d)} = 2$$
$$\texttt{edge(c, a)} = 9$$

Fig. 3. A cost graph and the corresponding initial database

$\oplus \rightarrow \max$

$\otimes \rightarrow \cdot$

most likely path



$$\texttt{initial(a)} = 1 \quad \texttt{edge(c, d)} = 0.4$$
$$\texttt{edge(a, c)} = 0.2 \quad \texttt{edge(d, b)} = 0.2$$
$$\texttt{edge(a, d)} = 0.8 \quad \texttt{edge(d, c)} = 0.3$$
$$\texttt{edge(b, b)} = 0.9 \quad \texttt{edge(d, d)} = 0.5$$
$$\texttt{edge(c, a)} = 0.6$$

Fig. 4. A probabilistic graph and the corresponding initial database. With stopping probabilities made explicit, this would encode a Markov model.

[Figures: Cohen et al, ICLP 08]

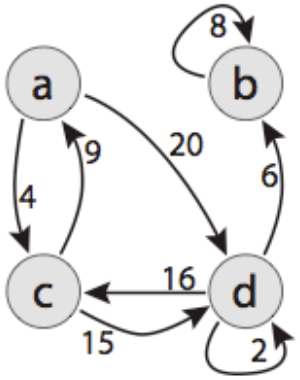
$$\texttt{reachable(Q)} \oplus\!= \texttt{initial(Q)}.$$

$$\texttt{reachable(Q)} \oplus\!= \texttt{reachable(P)} \otimes \texttt{edge(P, Q)}.$$

$\oplus \rightarrow \min$

$\otimes \rightarrow +$

shortest path



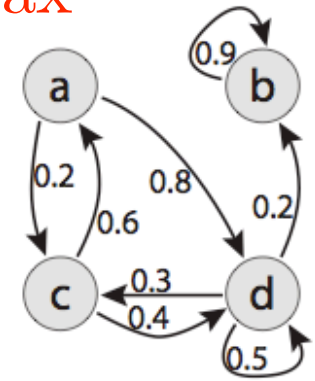
$\texttt{initial(a)} = 0$
$\texttt{edge(a, c)} = 4$
$\texttt{edge(a, d)} = 20$
$\texttt{edge(b, b)} = 8$
$\texttt{edge(c, a)} = 9$
$\texttt{edge(c, d)} = 15$
$\texttt{edge(d, b)} = 6$
$\texttt{edge(d, c)} = 16$
$\texttt{edge(d, d)} = 2$

Fig. 3. A cost graph and the corresponding initial database

$\oplus \rightarrow \max$

$\otimes \rightarrow \cdot$

most likely path



$\texttt{initial(a)} = 1$
$\texttt{edge(a, c)} = 0.2$
$\texttt{edge(a, d)} = 0.8$
$\texttt{edge(b, b)} = 0.9$
$\texttt{edge(c, a)} = 0.6$
$\texttt{edge(c, d)} = 0.4$
$\texttt{edge(d, b)} = 0.2$
$\texttt{edge(d, c)} = 0.3$
$\texttt{edge(d, d)} = 0.5$

$\oplus \rightarrow +$

$\otimes \rightarrow \cdot$

PRISM

Fig. 4. A probabilistic graph and the corresponding initial database. With stopping probabilities made explicit, this would encode a Markov model.

[Figures: Cohen et al, ICLP 08]

aProbLog: algebraic Prolog

[Kimmig et al, AAAI 11]

aProbLog: algebraic Prolog

[Kimmig et al, AAAI 11]

- Prolog + labeled facts

aProbLog: algebraic Prolog

[Kimmig et al, AAAI 11]

- Prolog + labeled facts
- labels from commutative semiring
  (e.g. probabilities, costs, polynomials,
  Boolean functions, datastructures, ...)

aProbLog: algebraic Prolog

[Kimmig et al, AAAI 11]

- Prolog + labeled facts
- labels from commutative semiring
  (e.g. probabilities, costs, polynomials,
  Boolean functions, datastructures, ...)
- inference based on ProbLog principles

aProbLog: algebraic Prolog

[Kimmig et al, AAAI 11]

- Prolog + labeled facts
- labels from commutative semiring
  (e.g. probabilities, costs, polynomials,
  Boolean functions, datastructures, ...)
- inference based on ProbLog principles

	neutral sum	NSP
disjoint sum	**SAT**	**MPE**
DSP	**PROB**	**#SAT**

aProbLog: algebraic Prolog

[Kimmig et al, AAAI 11]

- Prolog + labeled facts
- labels from commutative semiring
  (e.g. probabilities, costs, polynomials,
  Boolean functions, datastructures, ...)
- inference based on ProbLog principles

one world -
several proofs

	neutral sum	NSP
disjoint sum	**SAT**	**MPE**
DSP	**PROB**	**#SAT**

aProbLog: algebraic Prolog

[Kimmig et al, AAAI 11]

- Prolog + labeled facts
- labels from commutative semiring
  (e.g. probabilities, costs, polynomials,
  Boolean functions, datastructures, ...)
- inference based on ProbLog principles

one proof - several worlds

one world -
several proofs

	neutral sum	NSP
disjoint sum	**SAT**	**MPE**
DSP	**PROB**	**#SAT**

Algebraic Prolog (aProbLog)

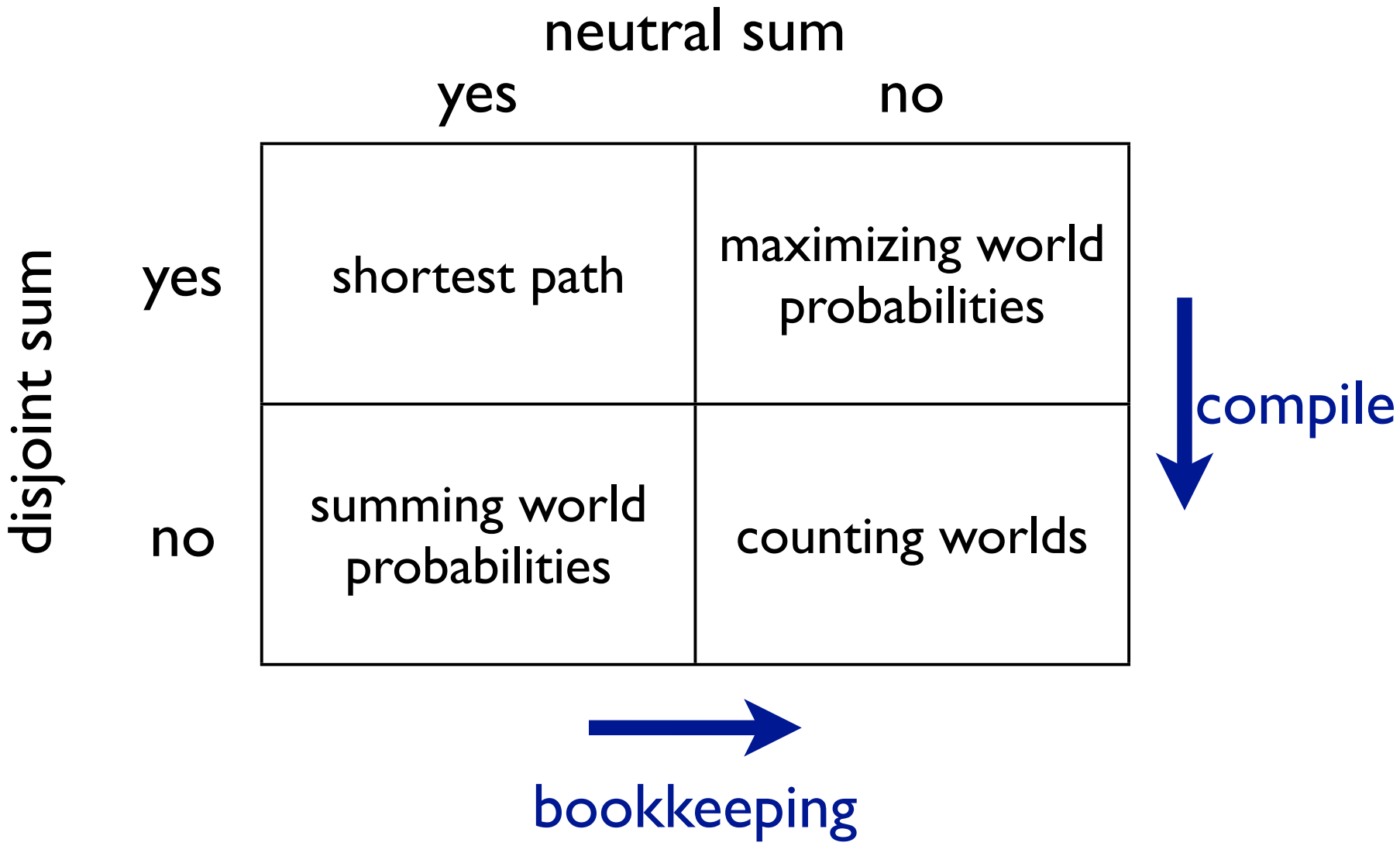
- commutative semiring $(A, \oplus, \otimes, e^{\oplus}, e^{\otimes})$
- algebraic ground literals
  $L(F) = \{f_1,...,f_n\} \cup \{\neg f_1,...,\neg f_n\}$
- background knowledge clauses
- labeling function $\alpha: L(F) \rightarrow A$

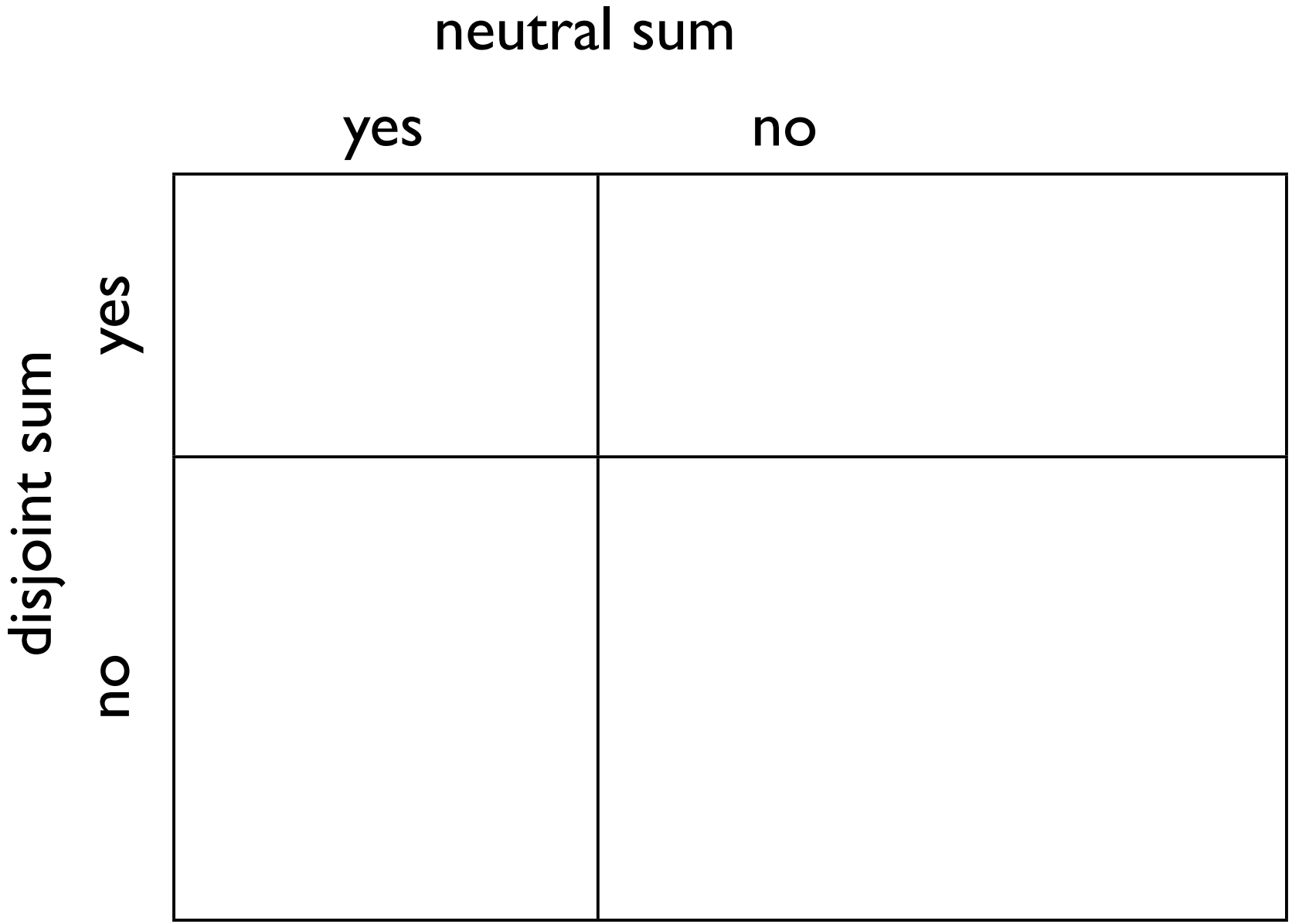
Algebraic Prolog (aProbLog)

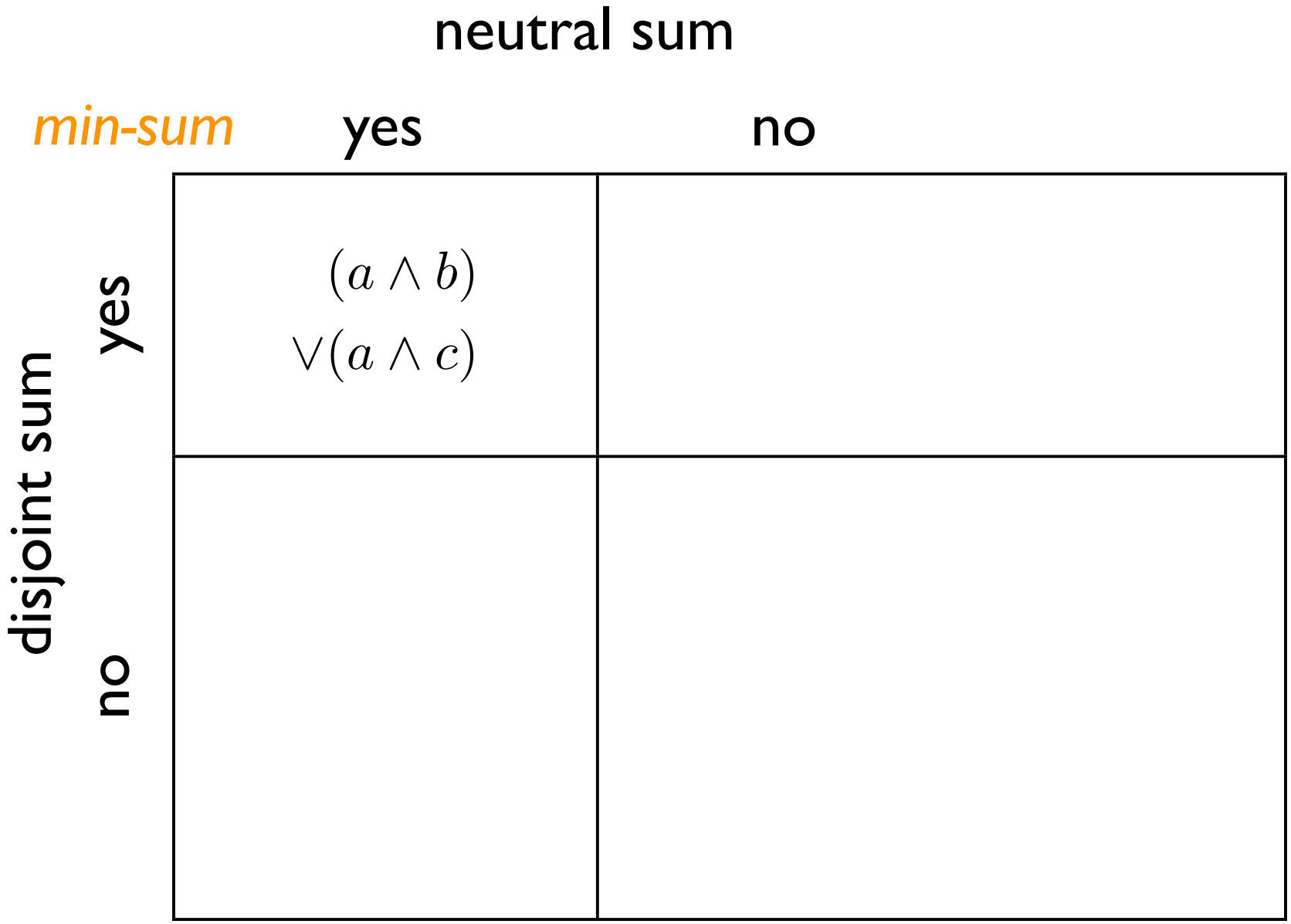
- commutative semiring $(A, \oplus, \otimes, e^{\oplus}, e^{\otimes})$
- algebraic ground literals
  $L(F) = \{f_1,...,f_n\} \cup \{\neg f_1,...,\neg f_n\}$
- background knowledge clauses
- labeling function $\alpha: L(F) \rightarrow A$

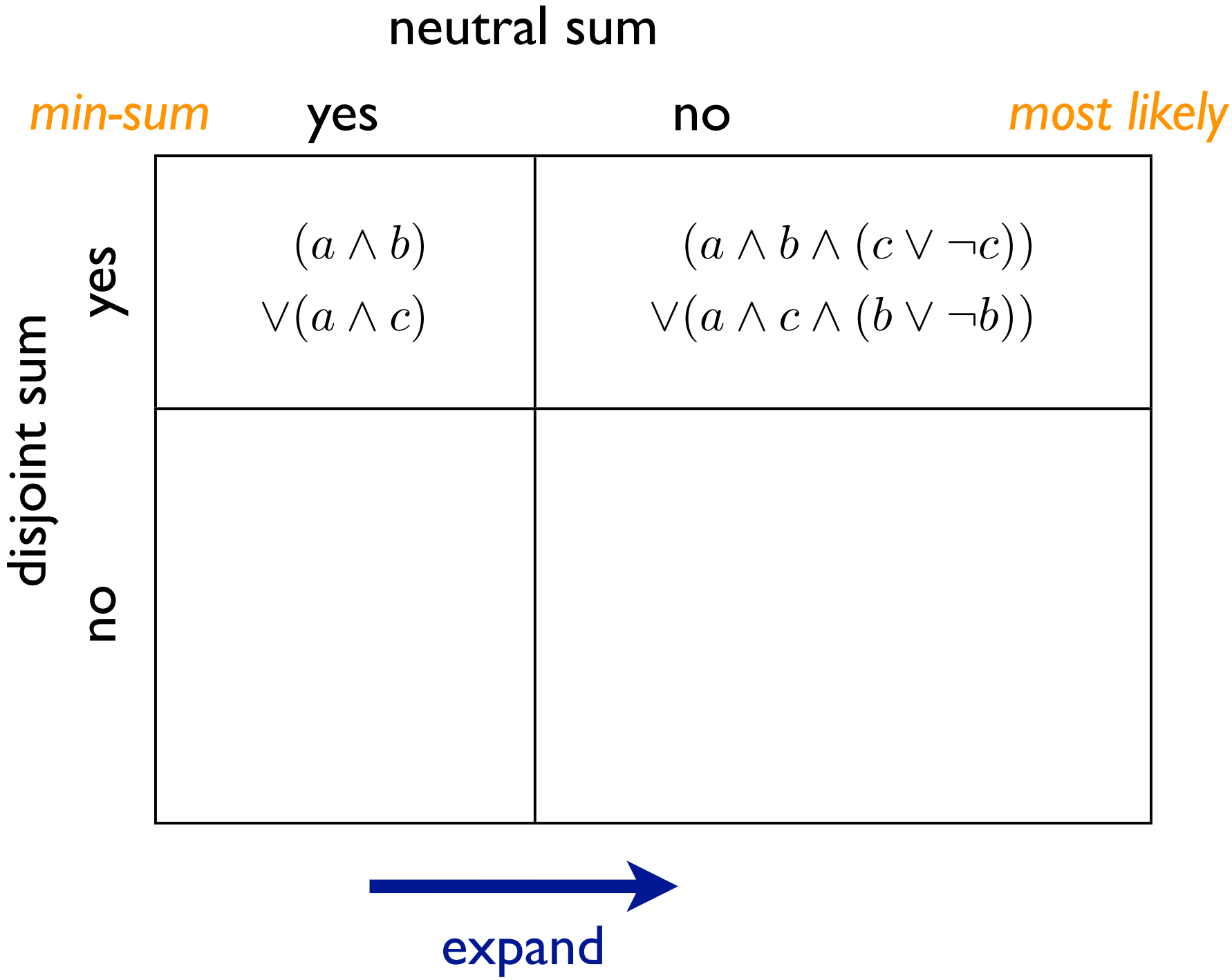
$$L(\text{query}) = \bigoplus_{\text{worlds}} \bigotimes_{\text{literals}} \alpha(l)$$

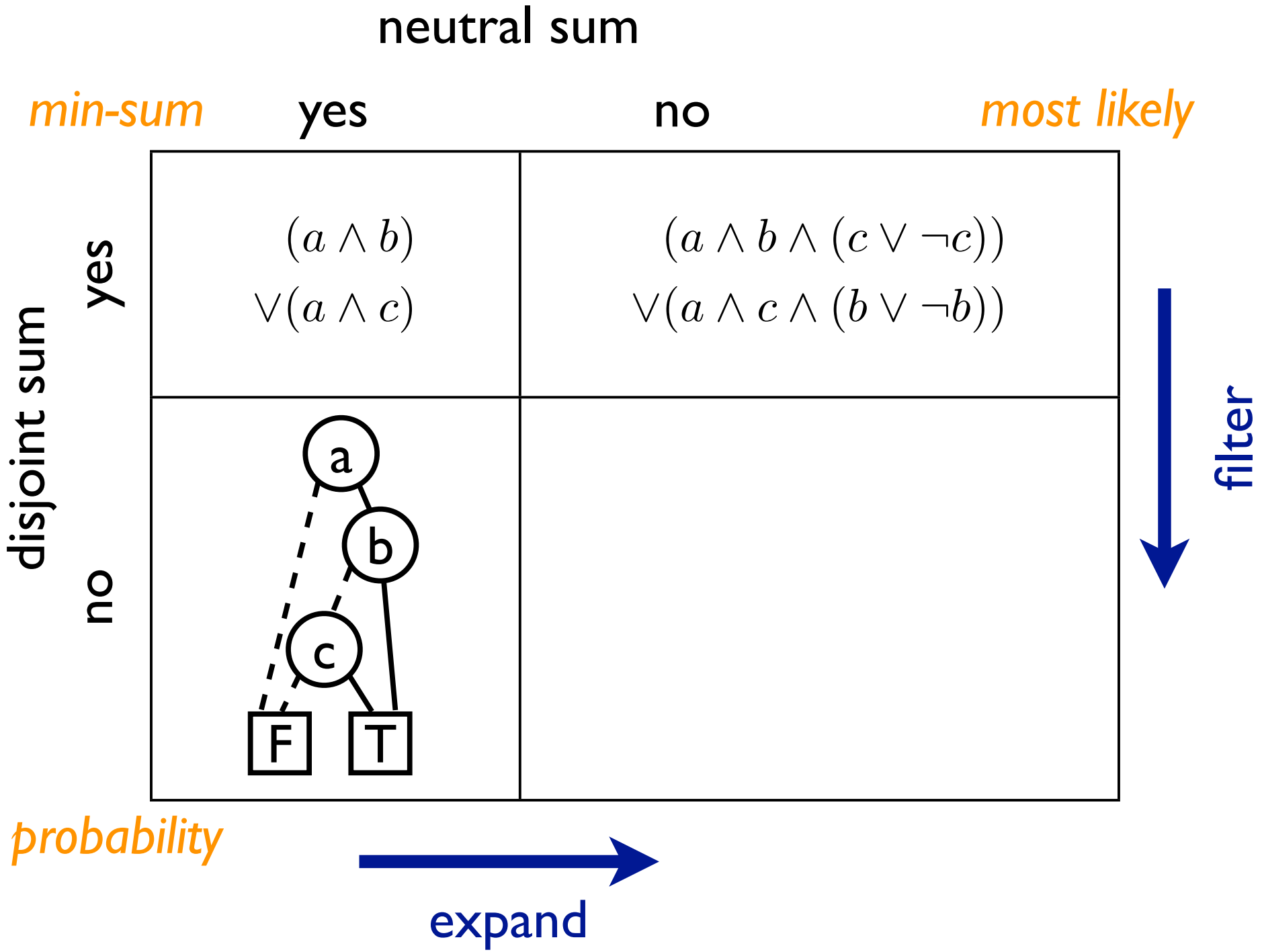
Inference settings

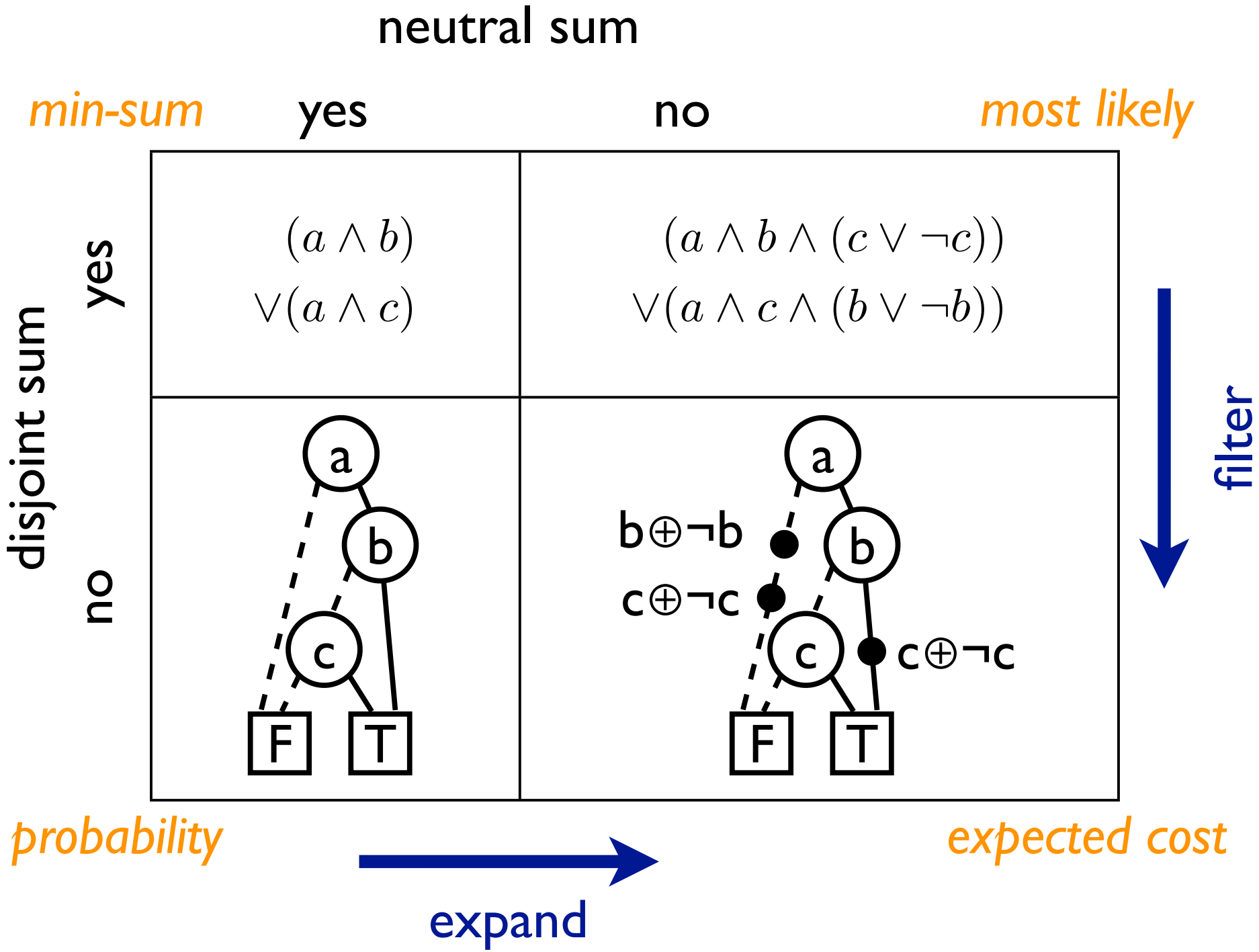



neutral sum
yes
no
disjoint sum
yes
no


neutral sum
min-sum
yes
no
disjoint sum
yes
no
$(a \wedge b)$
$\vee(a \wedge c)$


neutral sum
min-sum
yes
no
most likely
disjoint sum
yes
no
$(a \wedge b)$
$\vee(a \wedge c)$
$(a \wedge b \wedge (c \vee \neg c))$
$\vee(a \wedge c \wedge (b \vee \neg b))$
expand


neutral sum
min-sum
yes
no
most likely
disjoint sum
yes
no
$(a \wedge b)$
$\vee(a \wedge c)$
$(a \wedge b \wedge (c \vee \neg c))$
$\vee(a \wedge c \wedge (b \vee \neg b))$
a
b
c
F
T
filter
probability
expand


neutral sum
min-sum
yes
no
most likely
disjoint sum
yes
no
$(a \wedge b)$
$\vee(a \wedge c)$
$(a \wedge b \wedge (c \vee \neg c))$
$\vee(a \wedge c \wedge (b \vee \neg b))$
a
b
c
F
T
b⊕¬b
c⊕¬c
c⊕¬c
filter
probability
expected cost
expand

Approximate Inference

- Lower and upper bounds

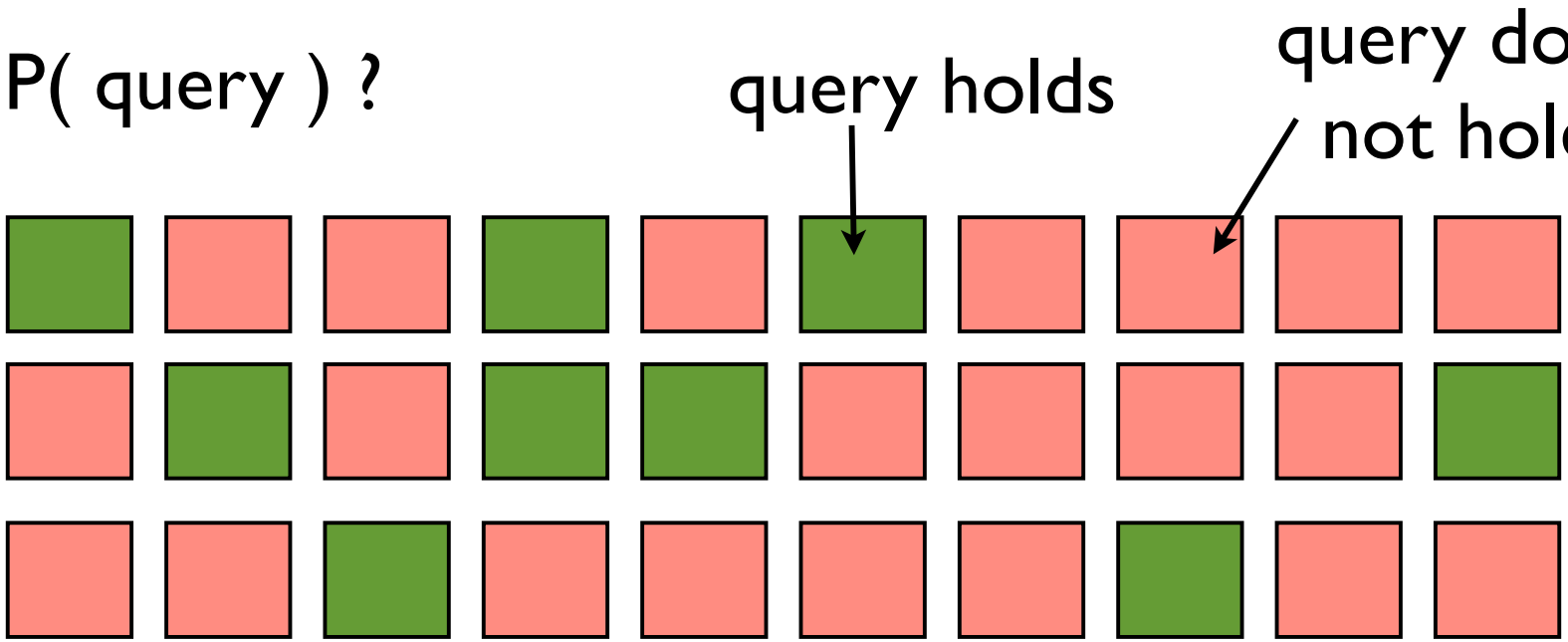
$$\phi_L \models \phi \models \phi_U$$

$$P(\phi_L) \leq P(\phi) \leq P(\phi_U)$$

- Sampling

Sampling

- P(query) ?



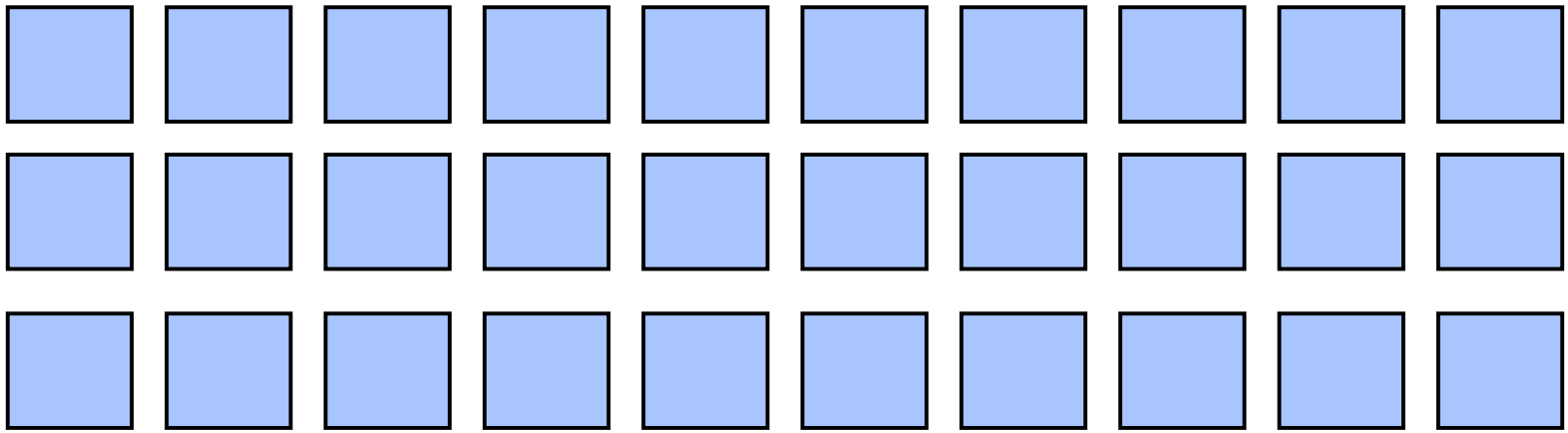
$$P(\text{query}) \approx \frac{\#\text{ query holds}}{\#\text{ worlds sampled}}$$

Rejection Sampling

- P(query | evidence) ?

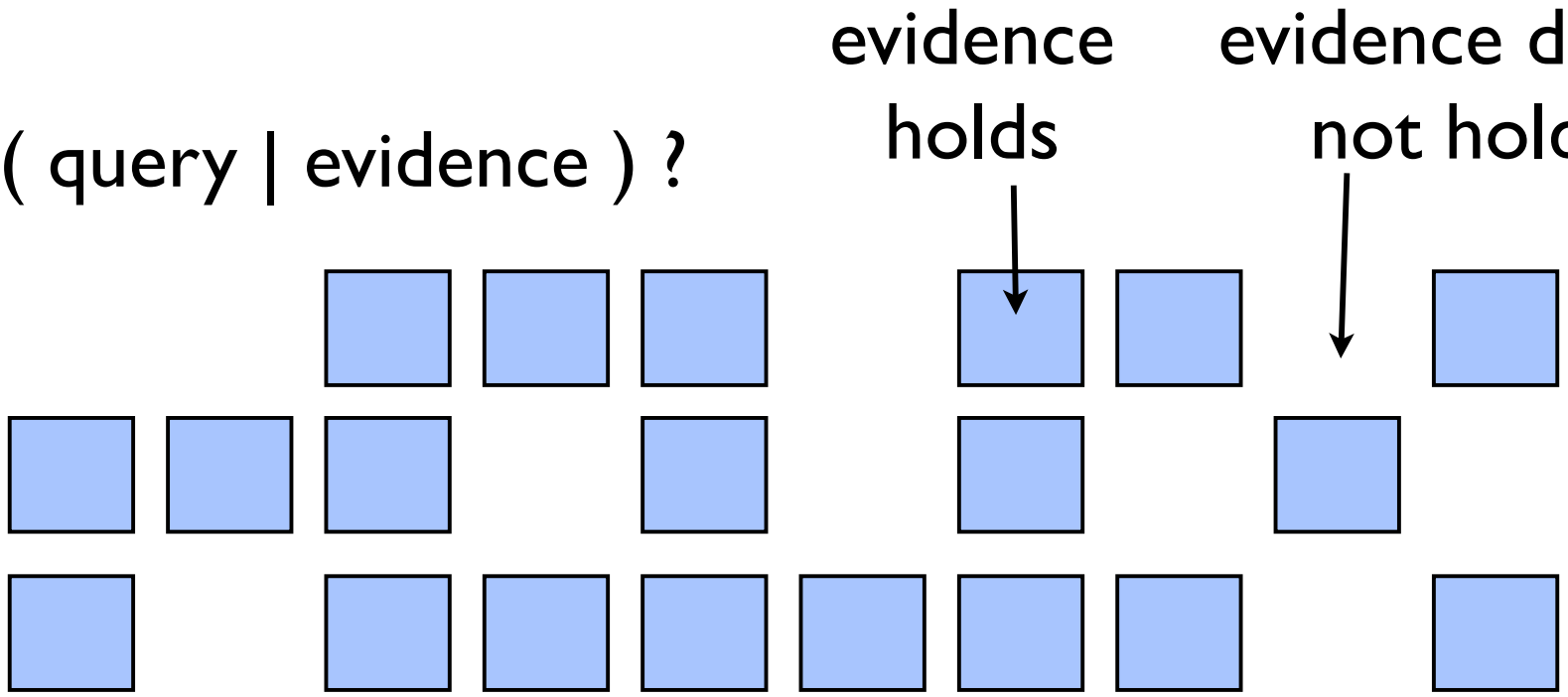
Rejection Sampling

- P(query | evidence) ?

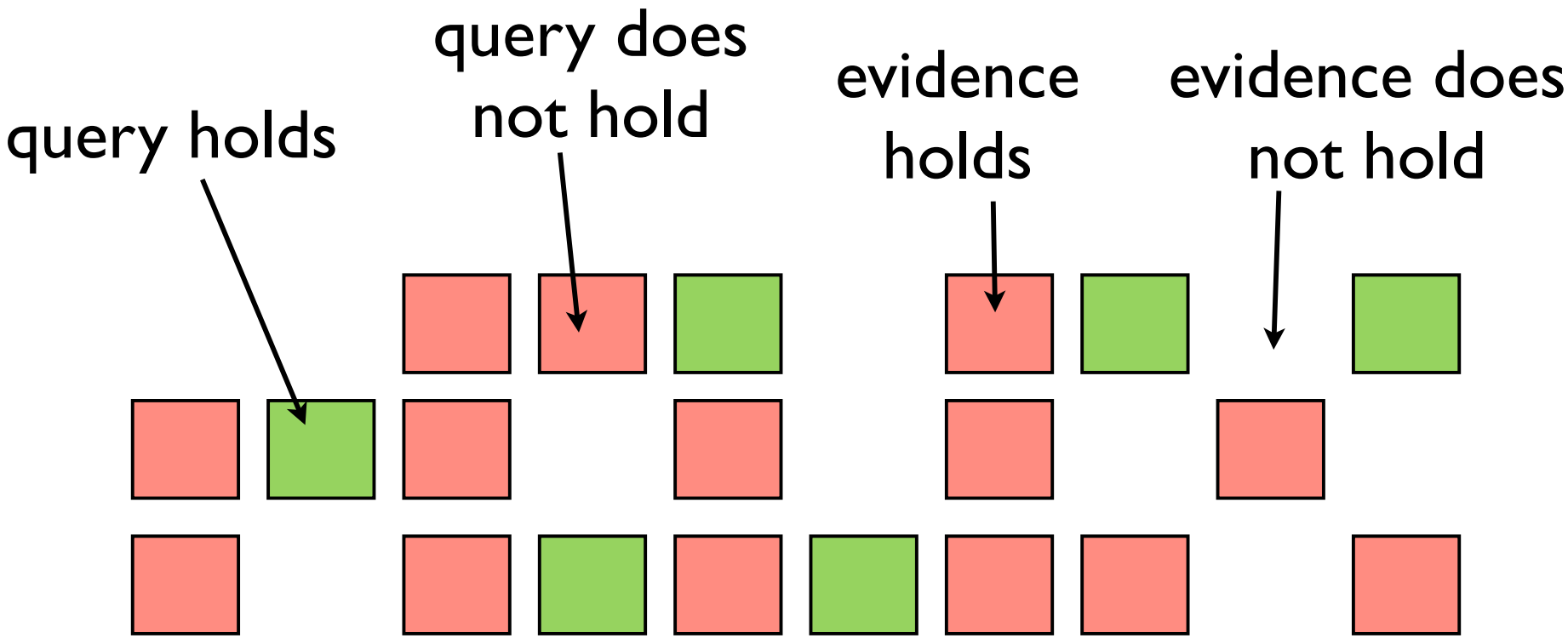


Rejection Sampling

- P(query | evidence) ?



Rejection Sampling



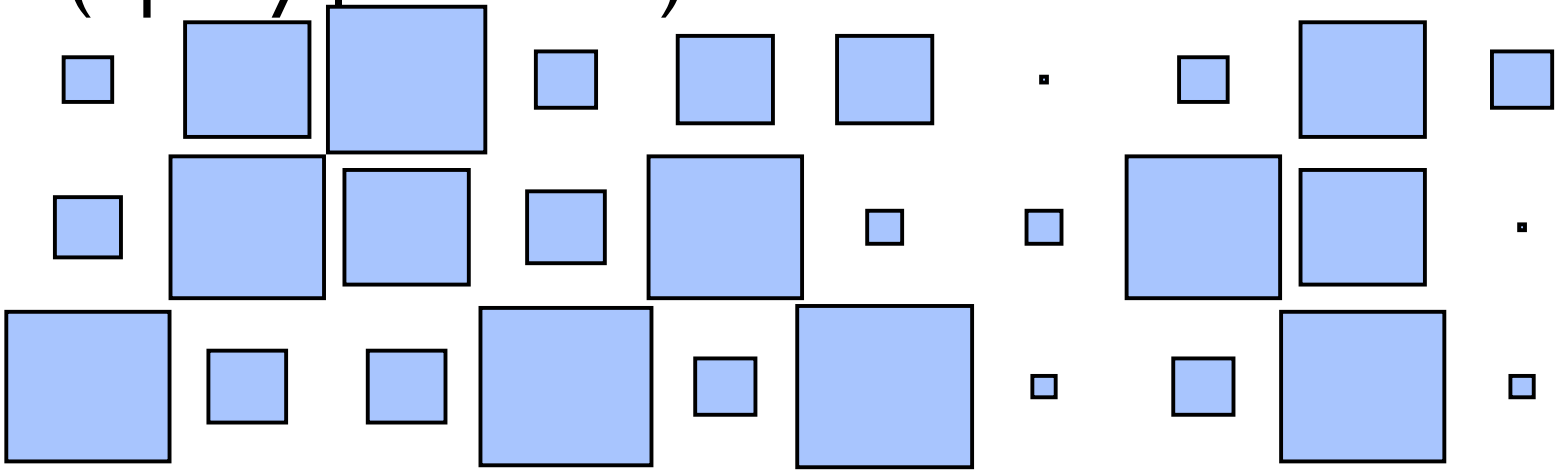
$$P(\text{query} \mid \text{evidence}) \approx \frac{\#\ \text{query \& evidence holds}}{\#\ \text{evidence holds}}$$

Likelihood Weighting

- P(query | evidence) ?

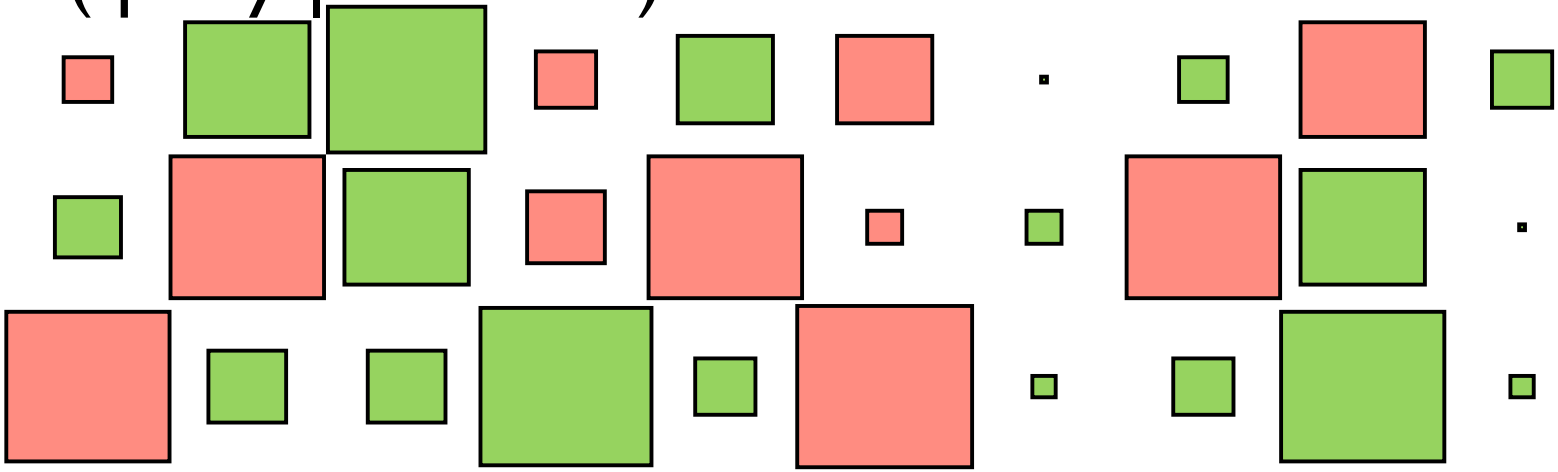
Likelihood Weighting

- P(query | evidence) ?



Likelihood Weighting

- P(query | evidence) ?



Markov Chain Monte Carlo (MCMC)

- Generate next sample by modifying current one
- Most common inference approach for PP languages such as Church, BLOG, ...
- Also considered for PRISM and ProbLog

Key challenges:
- how to propose next sample
- how to handle evidence

Roadmap

- Modeling
- Inference
- Learning & Dynamics
- Advanced Inference and KBMC

... with some detours on the way

Part III : Learning and Dynamics

Parameter Learning

e.g., webpage classification model

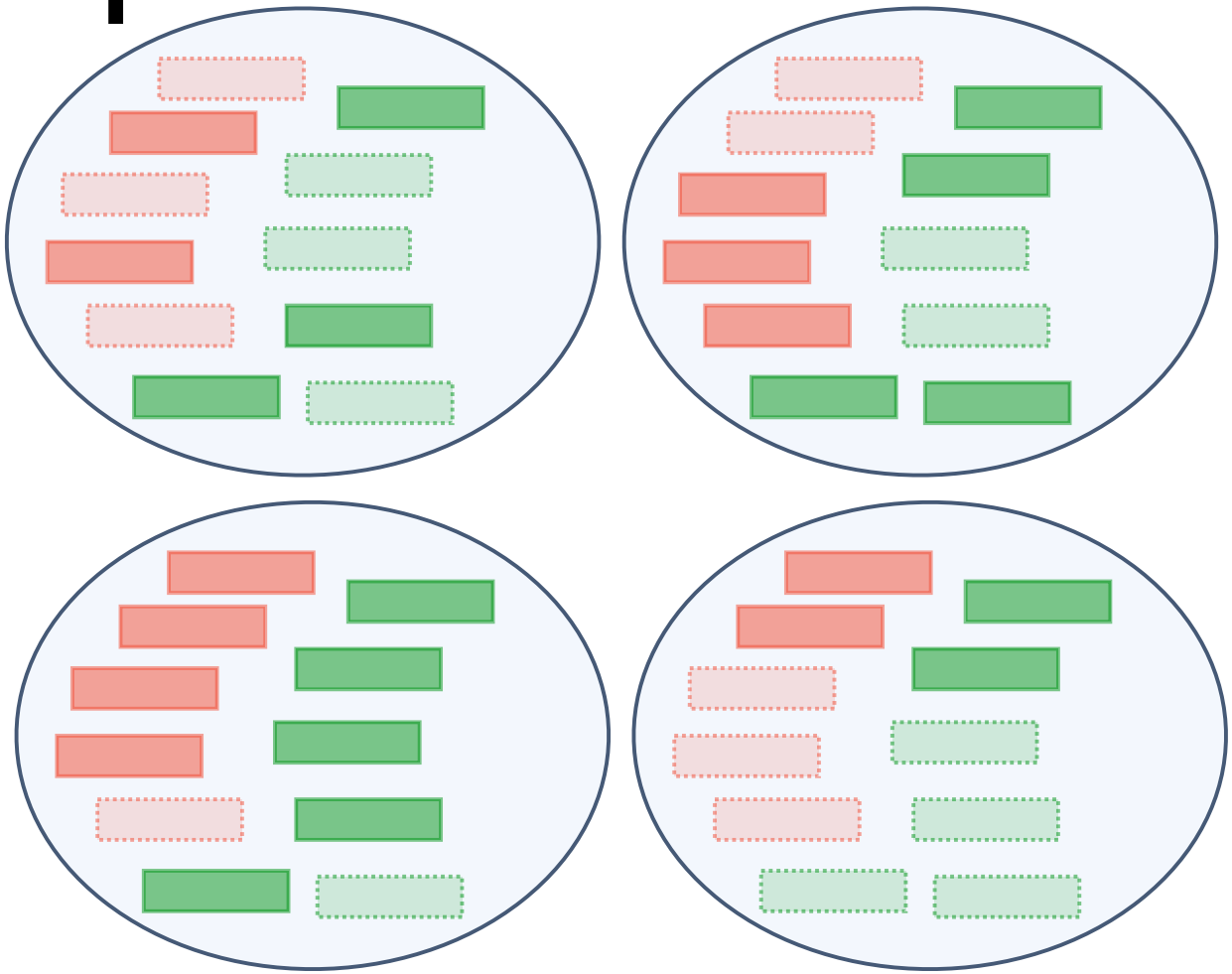
for each ***CLASS1***, ***CLASS2*** and each ***WORD***

?? :: link_class(Source,Target,***CLASS1***,***CLASS2***).
?? :: word_class(***WORD***,***CLASS***).

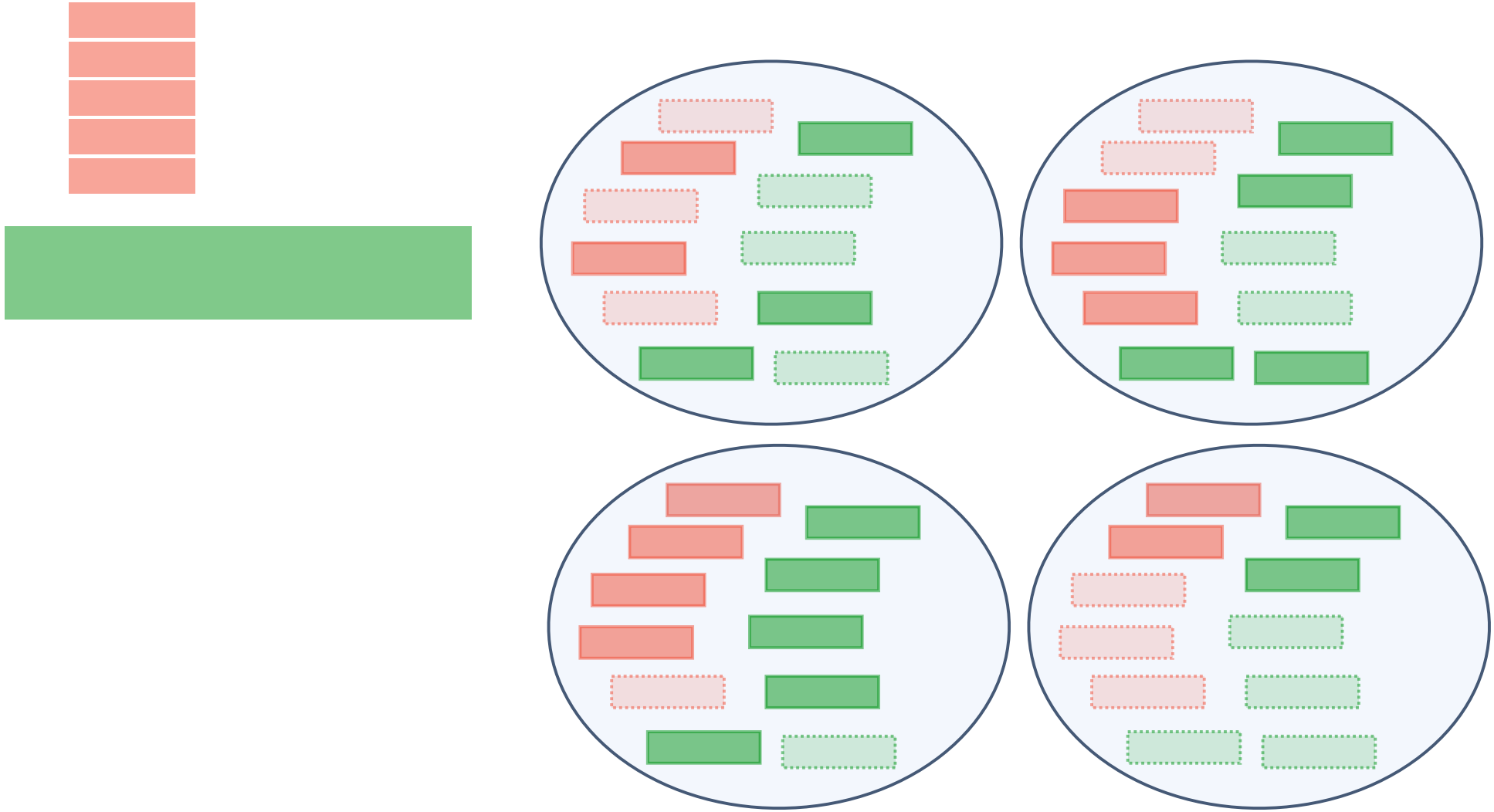
```
class(Page,C)    :-  has_word(Page,W), word_class(W,C).

class(Page,C)        :-  links_to(OtherPage,Page),
                         class(OtherPage,OtherClass),
                         link_class(OtherPage,Page,OtherClass,C).
```

Sampling Interpretations

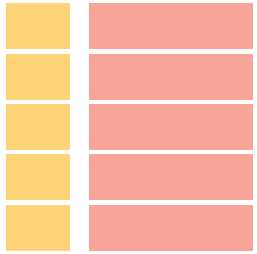
Sampling Interpretations

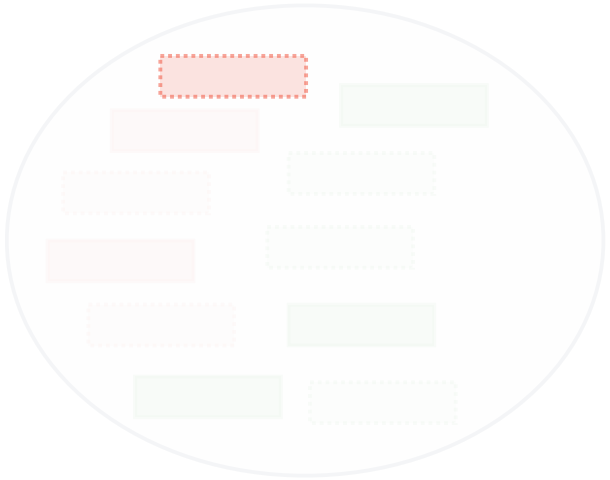


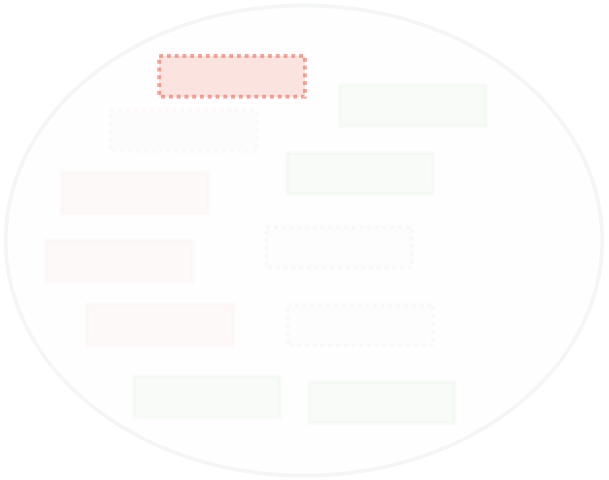
Parameter Estimation

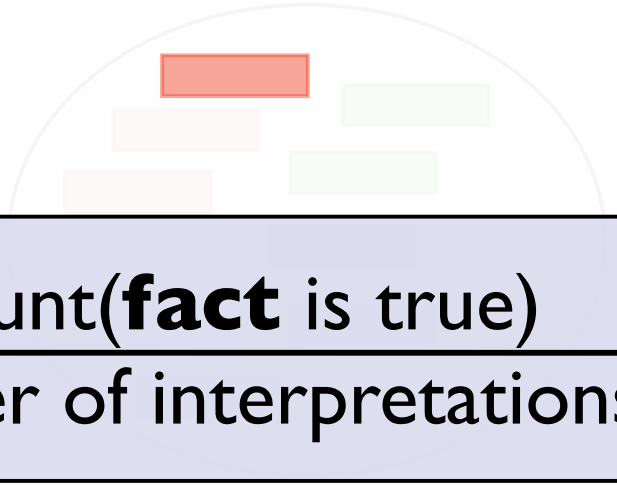


Parameter Estimation



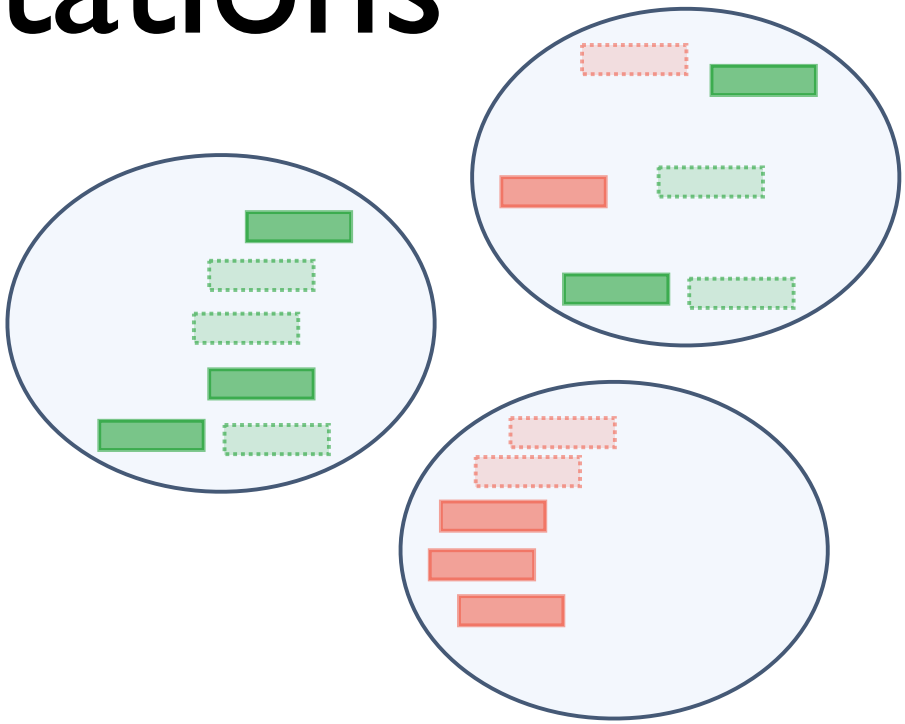






$$p(\textbf{fact}) = \frac{\text{count}(\textbf{fact} \text{ is true})}{\text{Number of interpretations}}$$

Learning from partial interpretations



- Not all facts observed
- Soft-EM
- use **expected count** instead of **count**
- **P(Q |E) -- conditional queries !**

 [Gutmann et al, ECML 11; Fierens et al, TPLP 14]

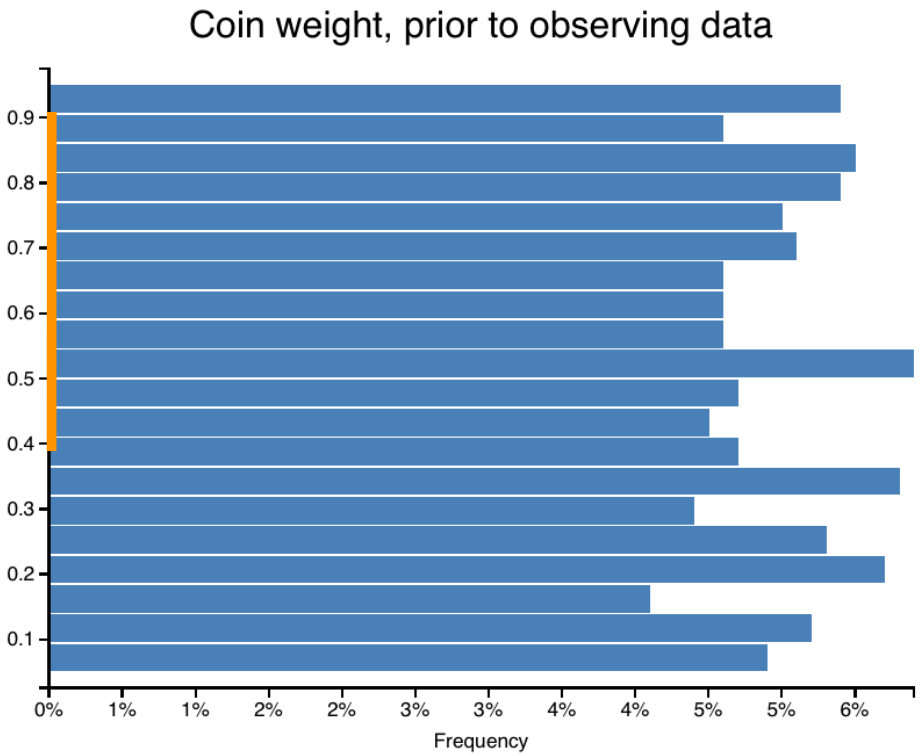
Bayesian Parameter Learning

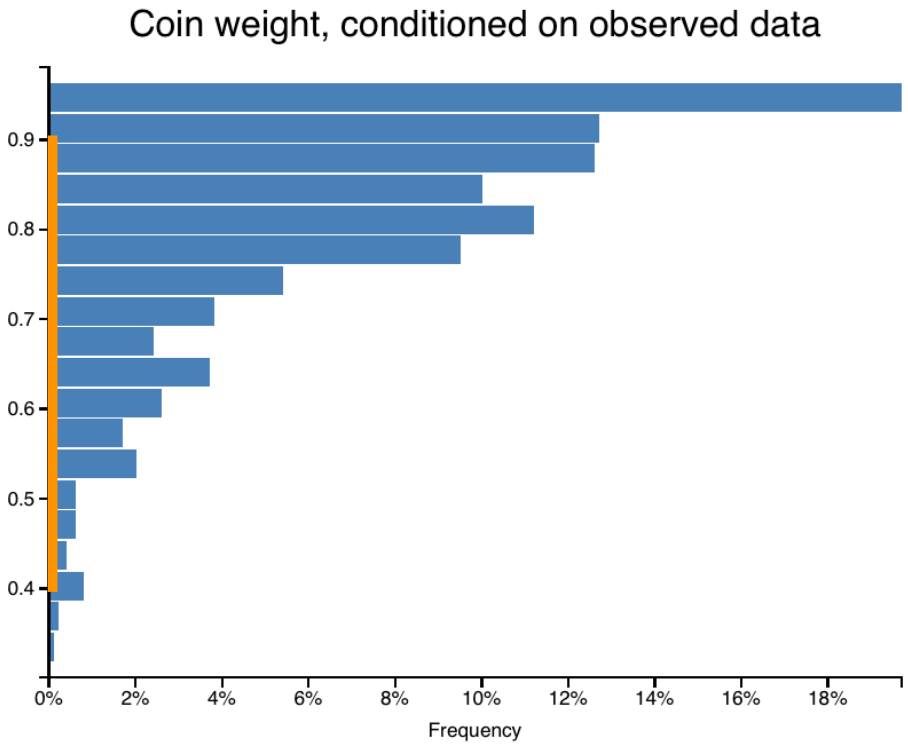
- Learning as inference (e.g., Church)
- Prior distributions for parameters
- Given data, find most likely parameter values

Example

[from probmods.org]

- Flipping a coin with unknown weight
- Prior: uniform distribution on [0,1]
- Observation: 5x heads in a row
- Sampling from Church model:





ProbLog Example

prior

```
0.05::weight(C,0.1); 0.2::weight(C,0.3); 0.5::weight(C,0.5);
                0.2::weight(C,0.7); 0.05::weight(C,0.9) <- coin(C).
```

```
Param::toss(_,Param,_).
heads(C,R) :- weight(C,Param),toss(C,Param,R).
tails(C,R) :- weight(C,Param),\+toss(C,Param,R).

data(C,[]).
data(C,[h|R]) :- heads(C,R), data(C,R).
data(C,[t|R]) :- tails(C,R), data(C,R).
```

```
coin(c1).
coin(c2).
param(0.1).
param(0.3).
param(0.5).
param(0.7).
param(0.9).
```

```
query(weight(C,X)) :- coin(C),param(X).
```

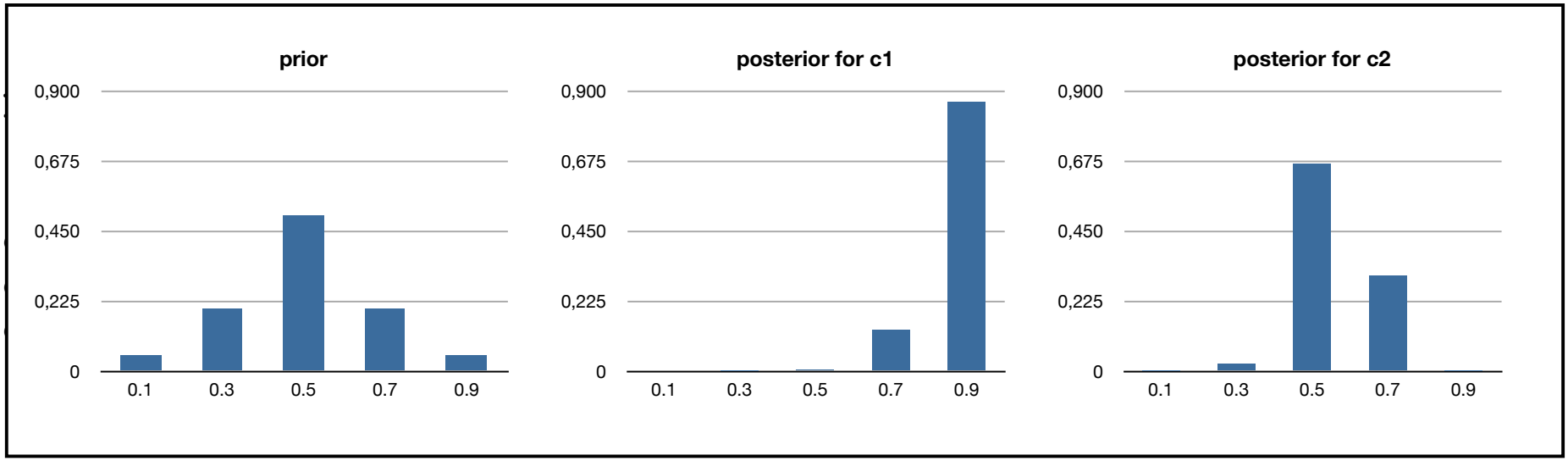
ask for posterior

```
evidence(data(c1,[h,h,h,h,h,h,h,h,h,h,h,h,h,h]),true).
evidence(data(c2,[h,t,h,h,h,h,h,t,t,h,t,t,h]),true).
```

data

ProbLog Example

```
0.05::weight(C,0.1); 0.2::weight(C,0.3); 0.5::weight(C,0.5);
                     0.2::weight(C,0.7); 0.05::weight(C,0.9) <- coin(C).
```



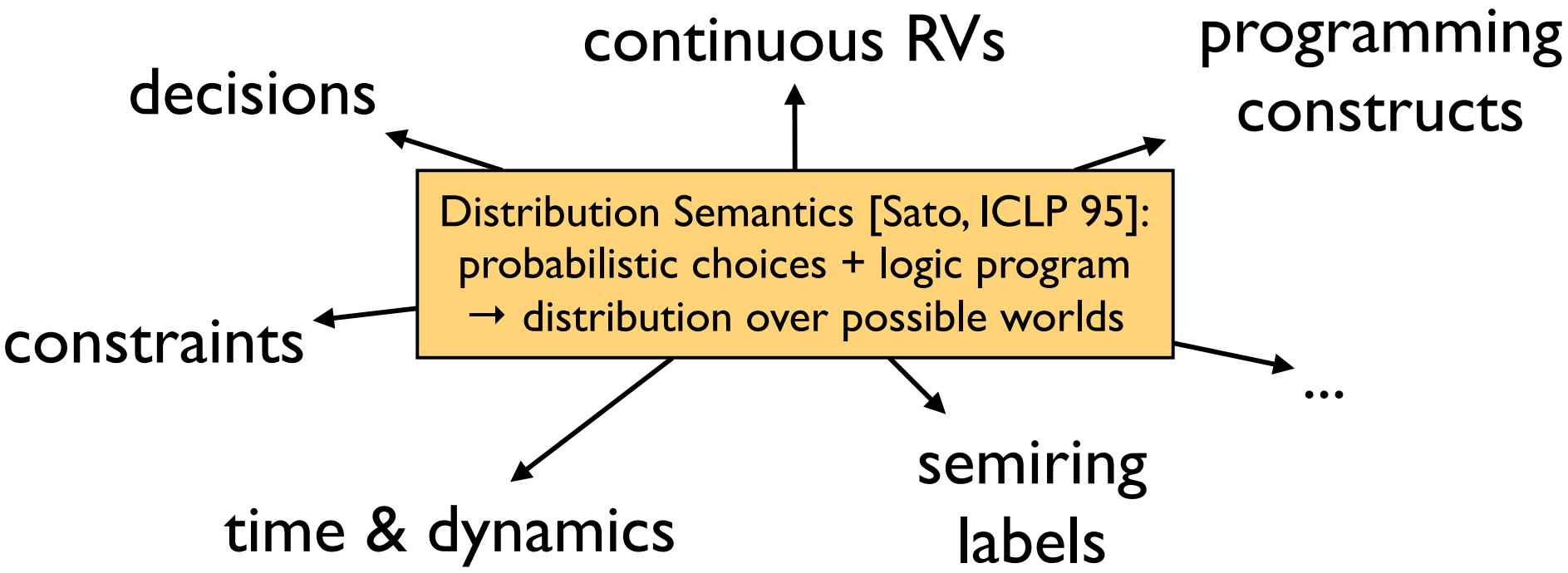

```
query(weight(C,X)) :- coin(C),param(X).

evidence(data(c1,[h,h,h,h,h,h,h,h,h,h,h,h,h,h]),true).
evidence(data(c2,[h,t,h,h,h,h,h,t,t,h,t,t,h]),true).
```

Roadmap

- Modeling
- Inference
- Learning & Dynamics
- Advanced Inference and KBMC

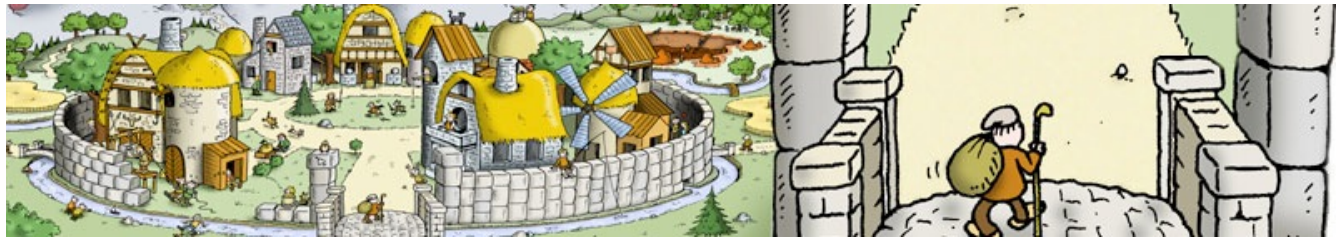
... with some detours on the way

Extensions of basic PLP



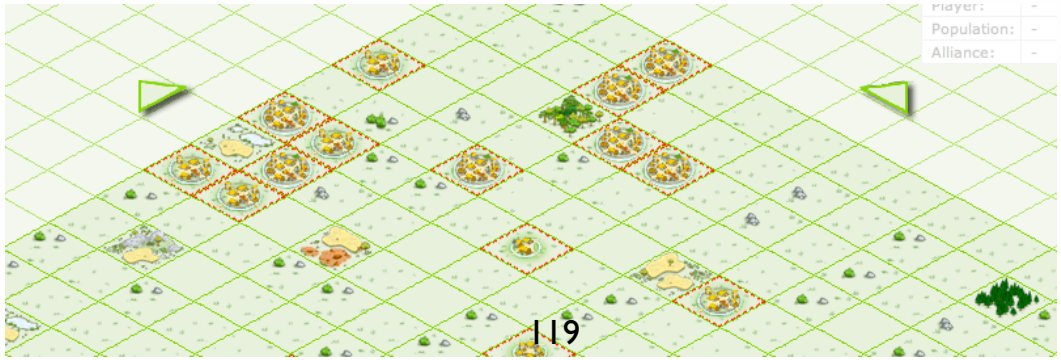
Dynamics

Dynamics: Evolving Networks



- *Travian*: A massively multiplayer real-time strategy game
  - Commercial game run by TravianGames GmbH
  - ~3.000.000 players spread over different "worlds"
  - ~25.000 players in one world

[Thon et al. ECML 08]



World Dynamics

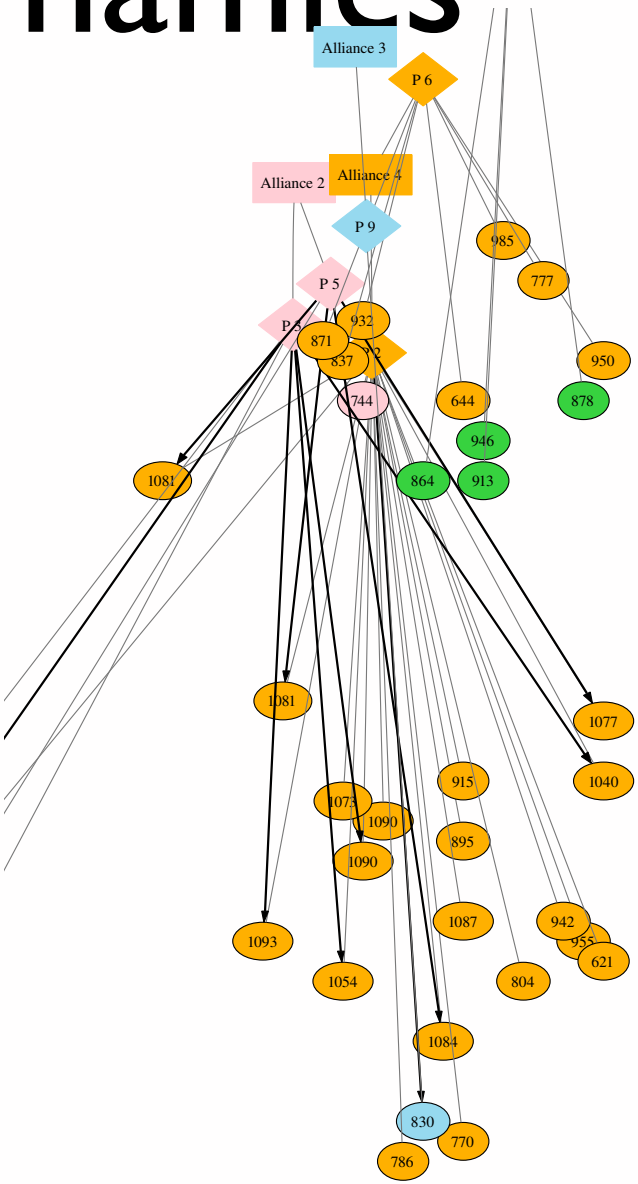
Fragment of world with

~10 alliances
~200 players
~600 cities

alliances color-coded

Can we build a model
of this world ?
Can we use it for playing
better ?

[Thon, Landwehr, De Raedt, ECML08]



World Dynamics

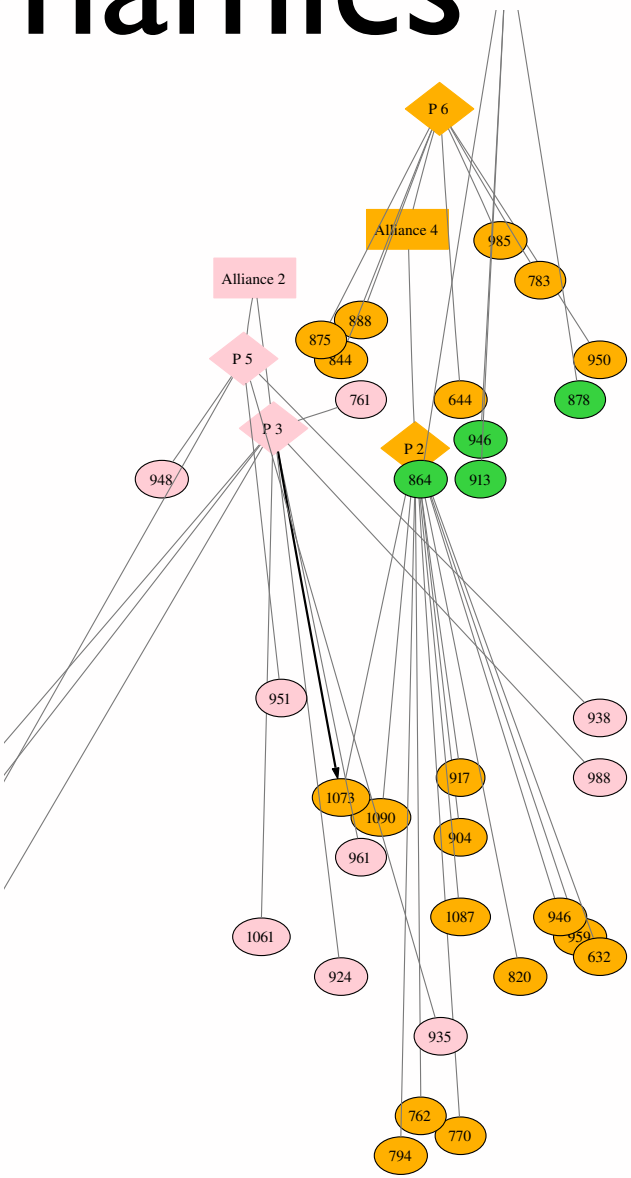
Fragment of world with

~10 alliances
~200 players
~600 cities

alliances color-coded

Can we build a model
of this world ?
Can we use it for playing
better ?

[Thon, Landwehr, De Raedt, ECML08]



World Dynamics

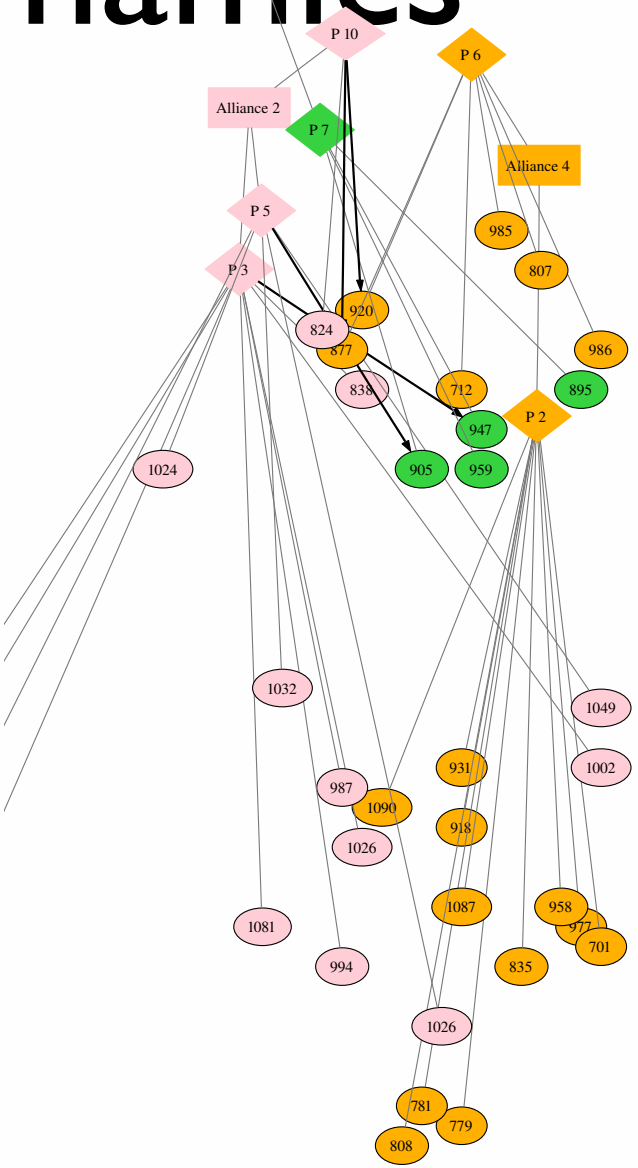
Fragment of world with

~10 alliances
~200 players
~600 cities

alliances color-coded

Can we build a model
of this world ?
Can we use it for playing
better ?

[Thon, Landwehr, De Raedt, ECML08]



World Dynamics

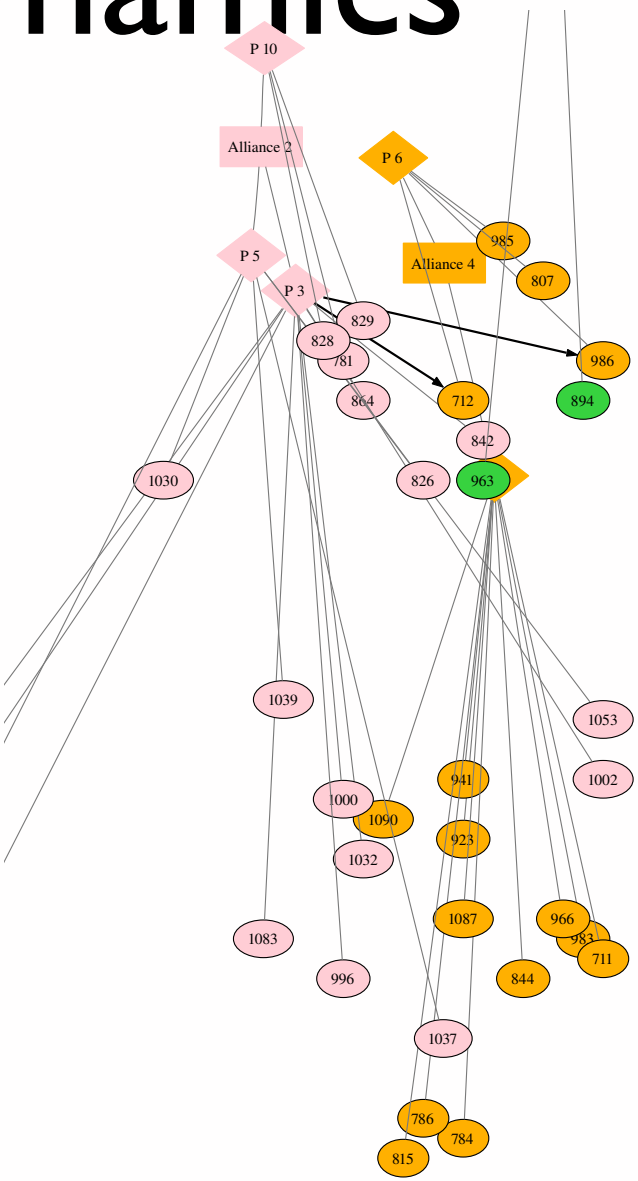
Fragment of world with

~10 alliances
~200 players
~600 cities

alliances color-coded

Can we build a model
of this world ?
Can we use it for playing
better ?

[Thon, Landwehr, De Raedt, ECML08]



World Dynamics

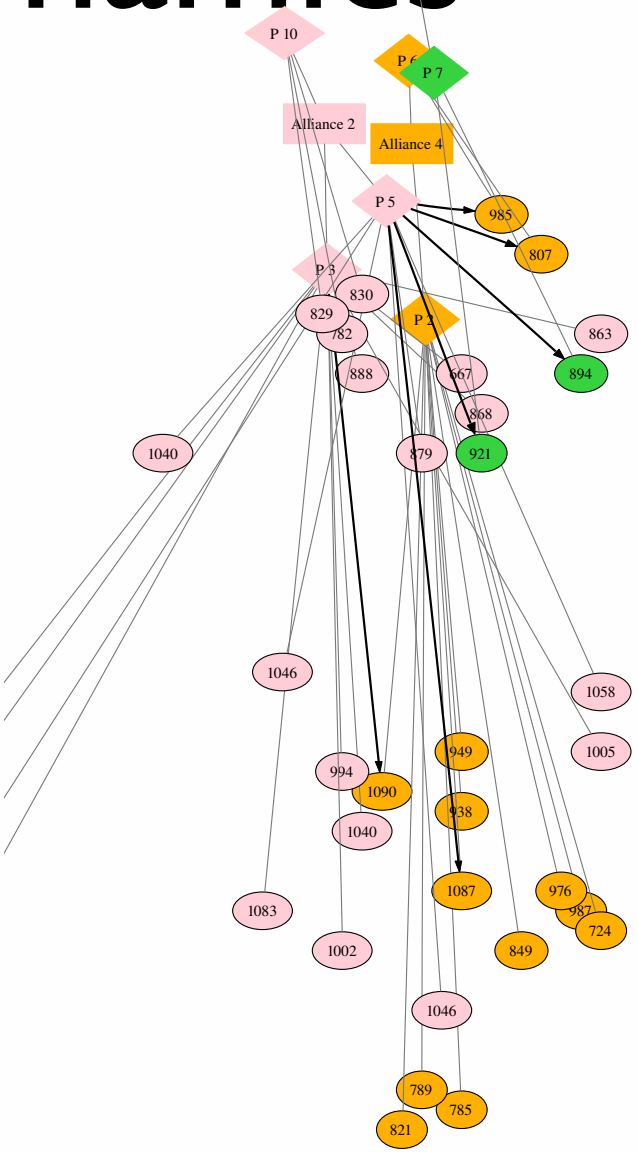
Fragment of world with

~10 alliances
~200 players
~600 cities

alliances color-coded

Can we build a model
of this world ?
Can we use it for playing
better ?

[Thon, Landwehr, De Raedt, ECML08]



World Dynamics

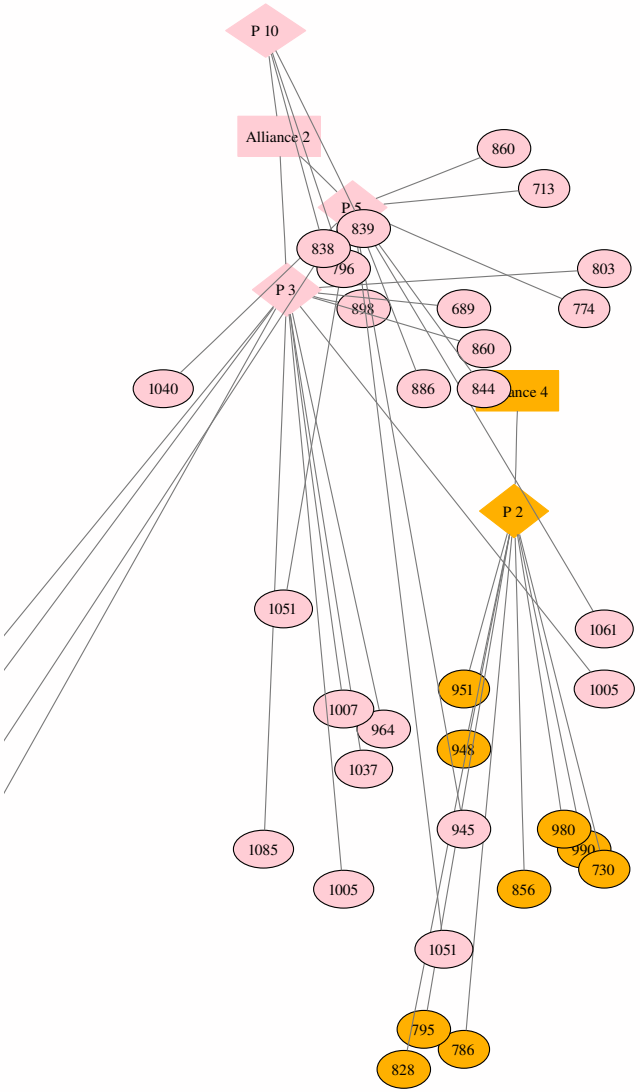
Fragment of world with

~10 alliances
~200 players
~600 cities

alliances color-coded

Can we build a model
of this world ?
Can we use it for playing
better ?

[Thon, Landwehr, De Raedt, ECML08]

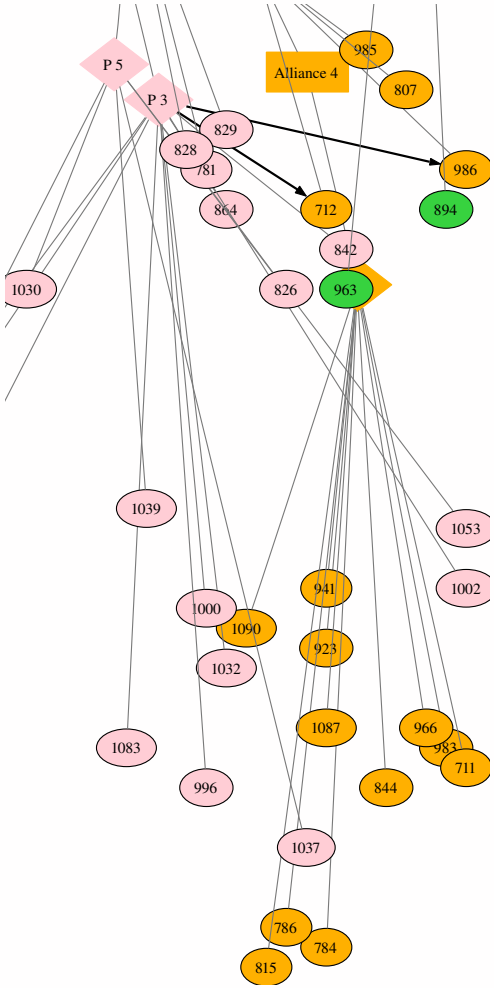


CPT-Rules

$$\underbrace{b_1, \ldots b_n}_{\text{cause}} \rightarrow \underbrace{h_1 : p_1 \vee \ldots \vee h_m : p_m}_{\text{effect}}$$

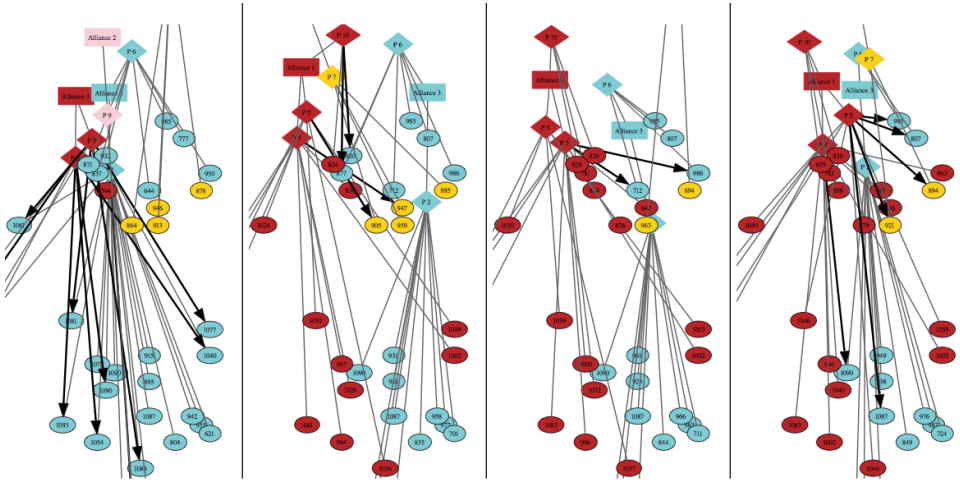
$$city(C, Owner), city(C2, Attacker), close(C, C2) \rightarrow$$
$$conquest(Attacker, C2) \ : \ p \ \vee \ nil \ : \ (1 - p)$$

conquer a city which is close
P(conquest(),Time+5) ?
learn parameters



Thon et al. MLJ 11

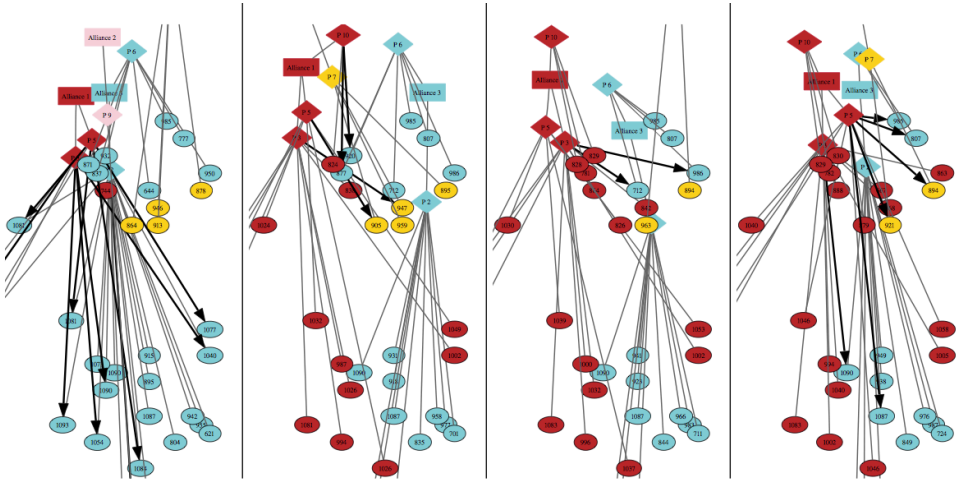
Causal Probabilistic Time-Logic (CPT-L)



how does the world change over time?

[Thon et al, MLJ 11]

Causal Probabilistic Time-Logic (CPT-L)



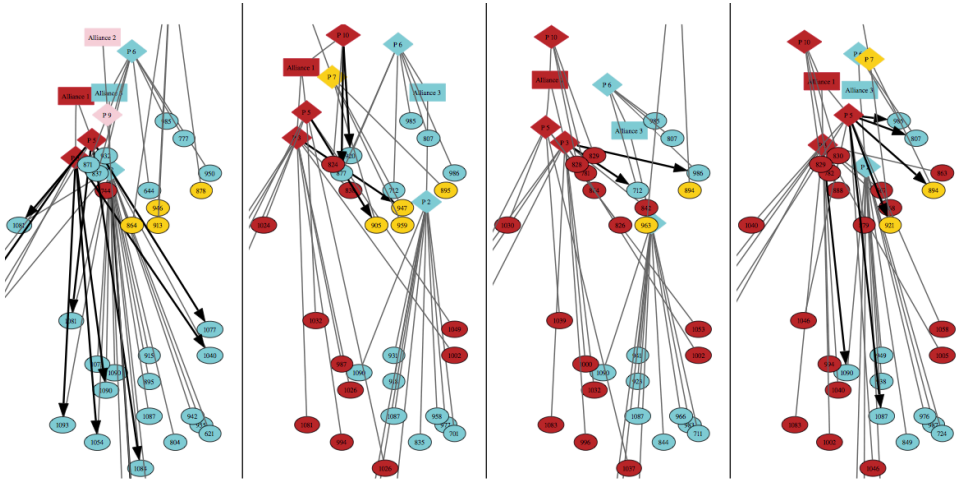
how does the world change over time?

```
0.4::conquest(Attacker,C); 0.6::nil <-
    city(C,Owner),city(C2,Attacker),close(C,C2).
```

if **cause** holds at time T

[Thon et al, MLJ 11]

Causal Probabilistic Time-Logic (CPT-L)



how does the world change over time?

one of the **effects** holds at time T+1

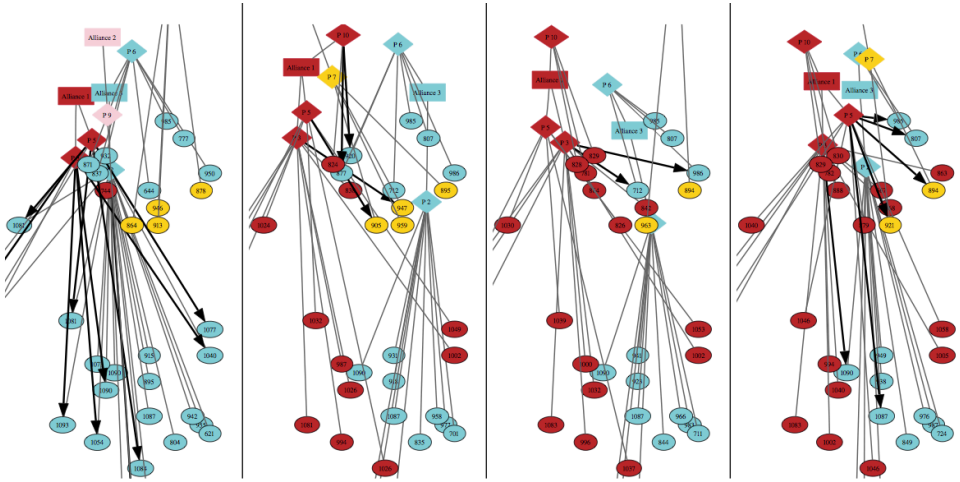
```
0.4::conquest(Attacker,C); 0.6::nil <-
```

```
city(C,Owner),city(C2,Attacker),close(C,C2).
```

if **cause** holds at time T

[Thon et al, MLJ 11]

Causal Probabilistic Time-Logic (CPT-L)



how does the world change over time?

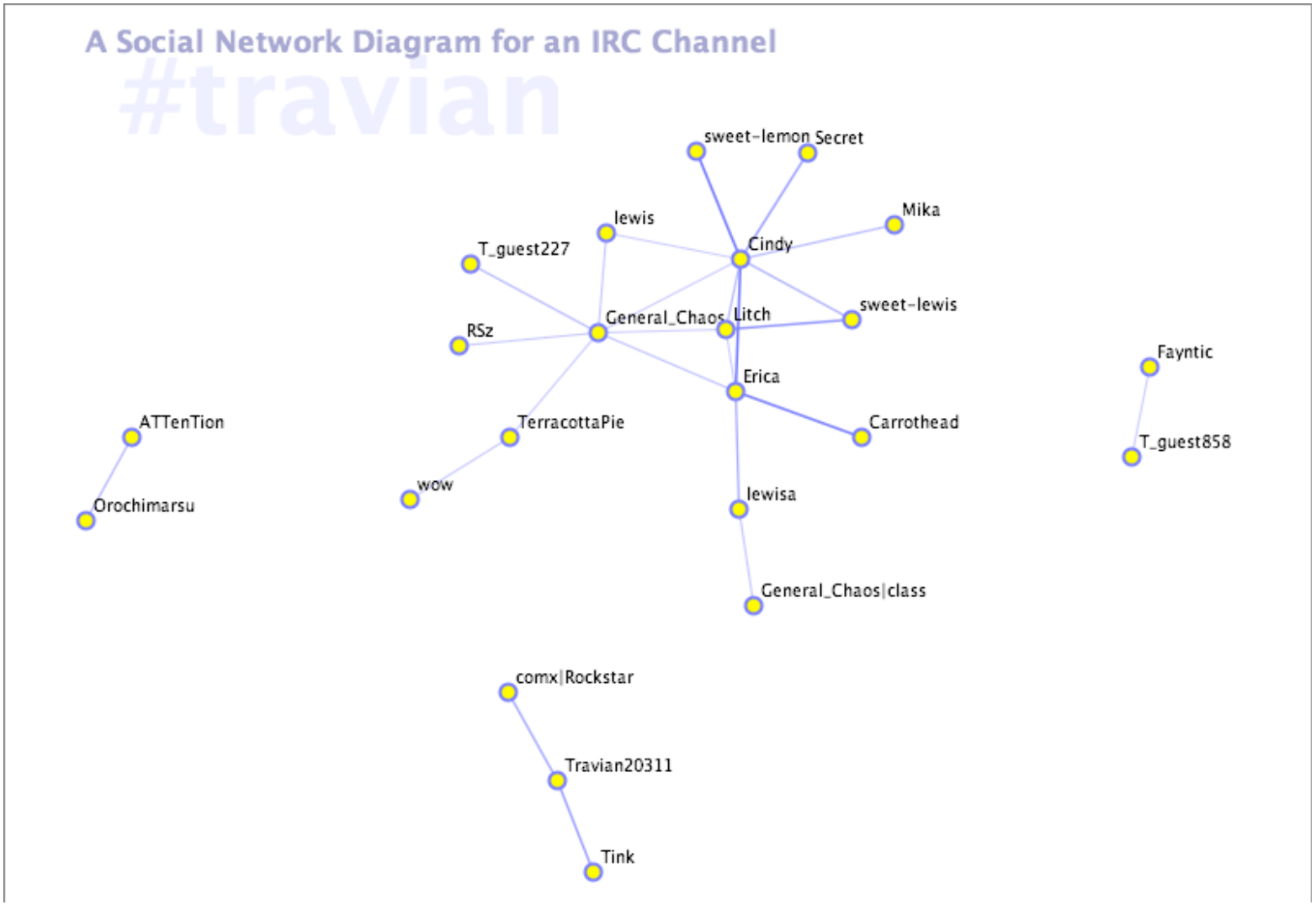
one of the **effects** holds at time T+1

```
0.4::conquest(Attacker,C); 0.6::nil <-
        city(C,Owner),city(C2,Attacker),close(C,C2).
```

if **cause** holds at time T

[Thon et al, MLJ 11]

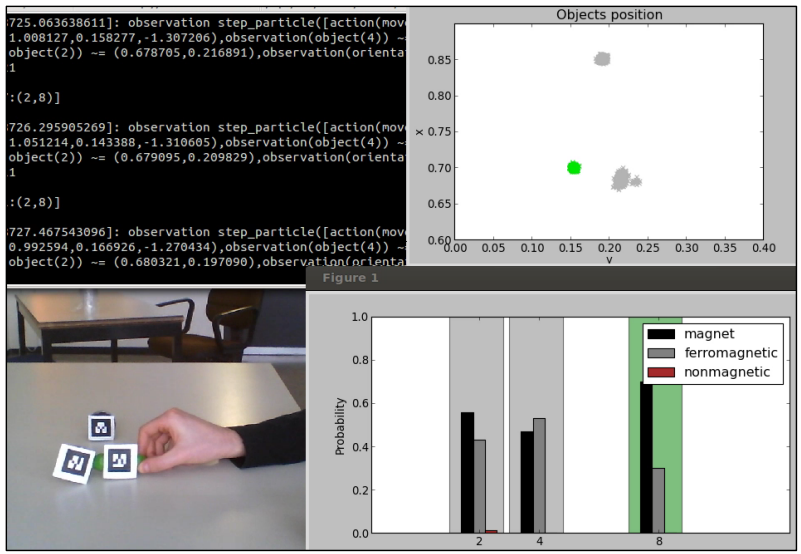
Social Network of Chats



Relational State Estimation over Time

Magnetism scenario

- object tracking
- category estimation from interactions



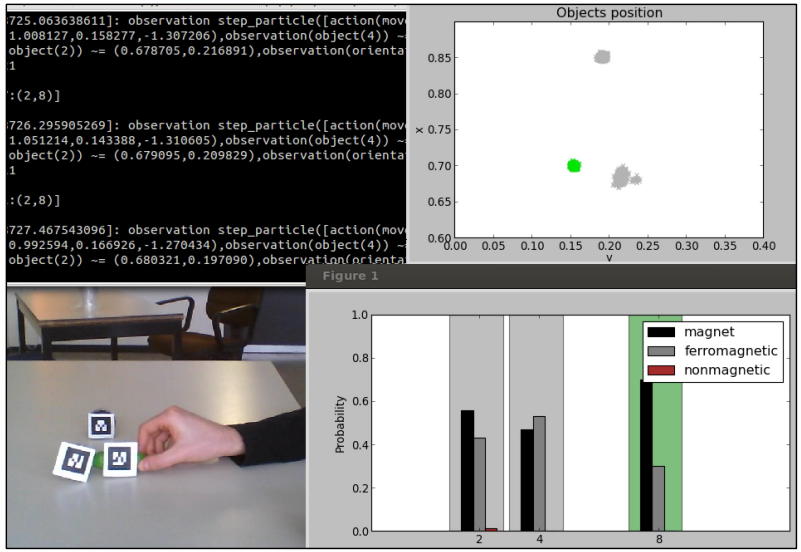
[Nitti et al, IROS 13]

Relational State Estimation over Time

Magnetism scenario

- object tracking
- category estimation from interactions

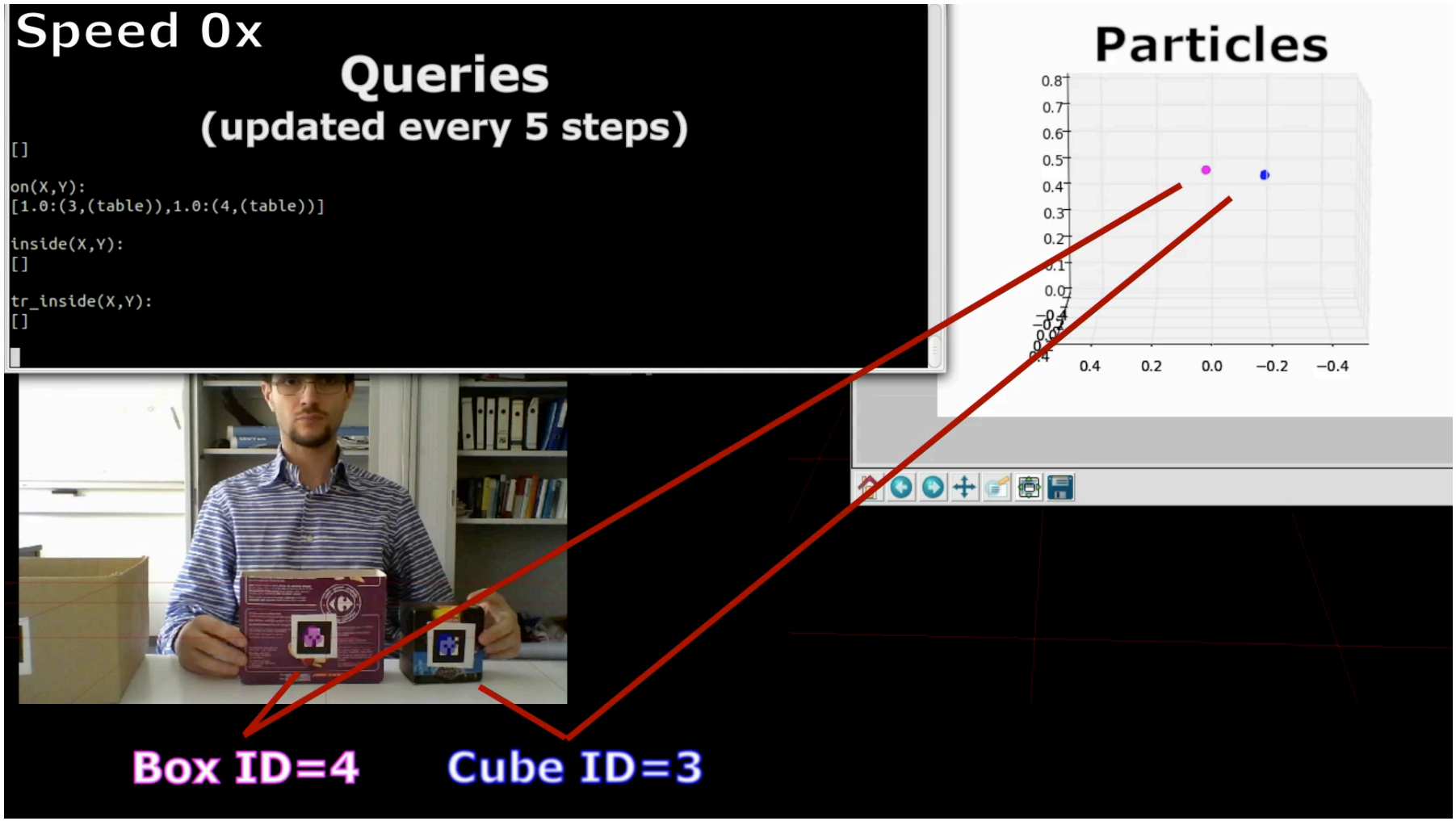
(a) cube on the box (b) cube inside the box (c) rotated box on a beige box (d) cube and box inside the beige box



Box scenario

- object tracking even when invisible
- estimate spatial relations

[Nitti et al, IROS 13]

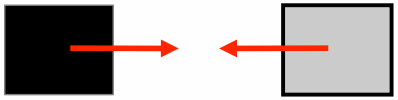

Speed 0x
Queries
(updated every 5 steps)
[]
on(X,Y):
[1.0:(3,(table)),1.0:(4,(table))]
inside(X,Y):
[]
tr_inside(X,Y):
[]
Particles
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.4
0.2
0.0
−0.2
−0.4
Box ID=4
Cube ID=3

IROS 13

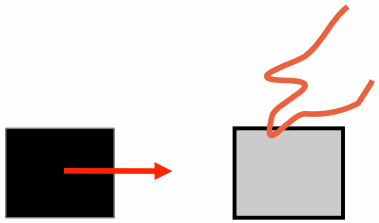
Magnetic scenario

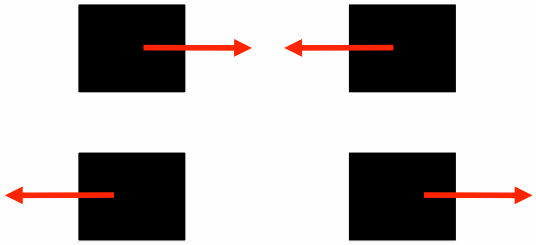
- 3 object types: magnetic, ferromagnetic, nonmagnetic

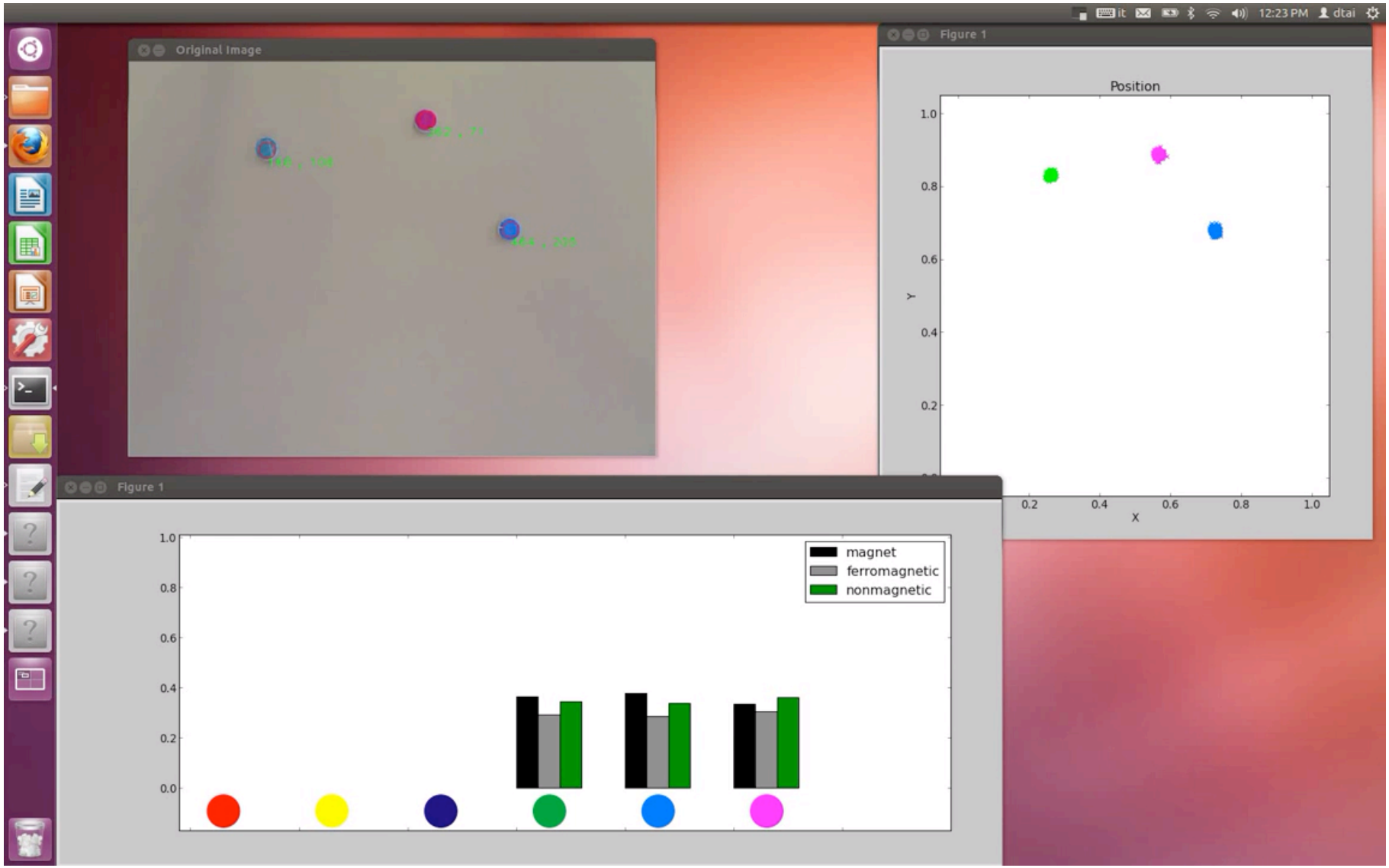
- Nonmagnetic objects do not interact
- A magnet and a ferromagnetic object attract each other



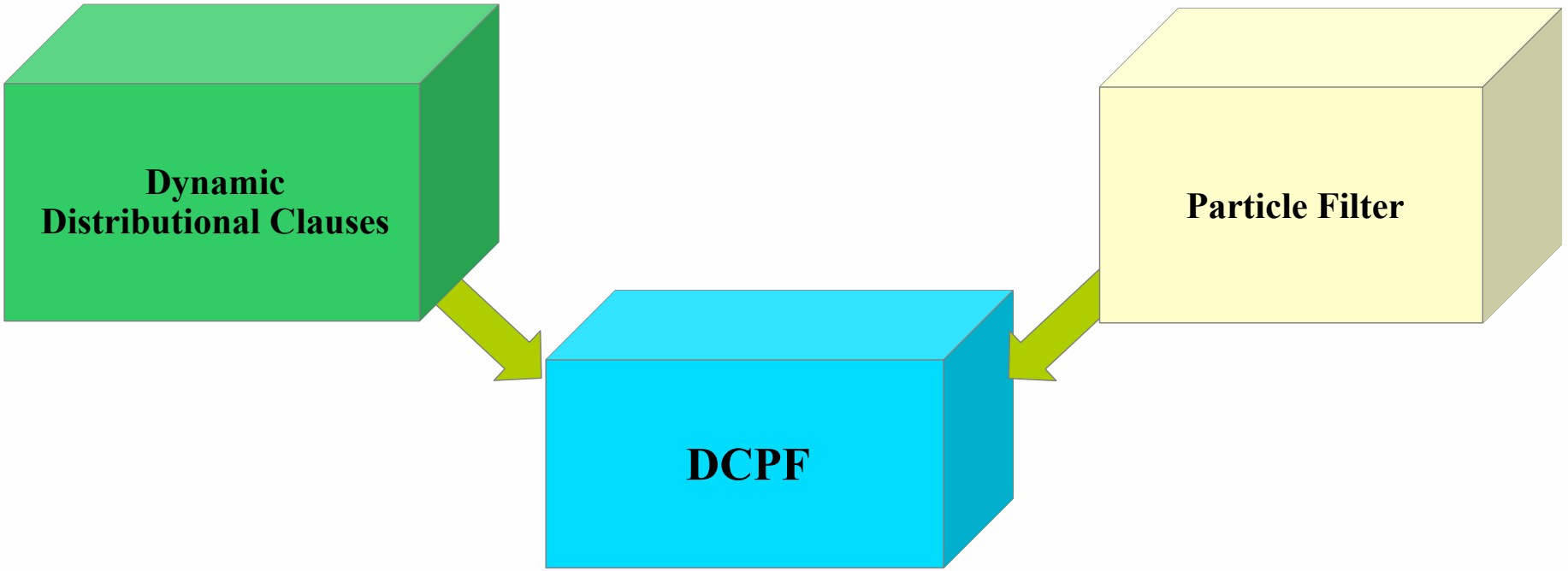
- Magnetic force that depends on the distance
- If an object is held magnetic force is compensated.






Original Image
Figure 1
Position
1.0
0.8
0.6
0.4
0.2
Y
0.2
0.4
0.6
0.8
1.0
X
Figure 1
1.0
0.8
0.6
0.4
0.2
0.0
magnet
ferromagnetic
nonmagnetic

DC Particle Filter (DCPF)



Goal

Flexible (relational) state representation

Fast inference (state estimation) in general models

"A particle filter for hybrid relational domains" IROS 2013
D. Nitti, T. De Laet, L. De Raedt

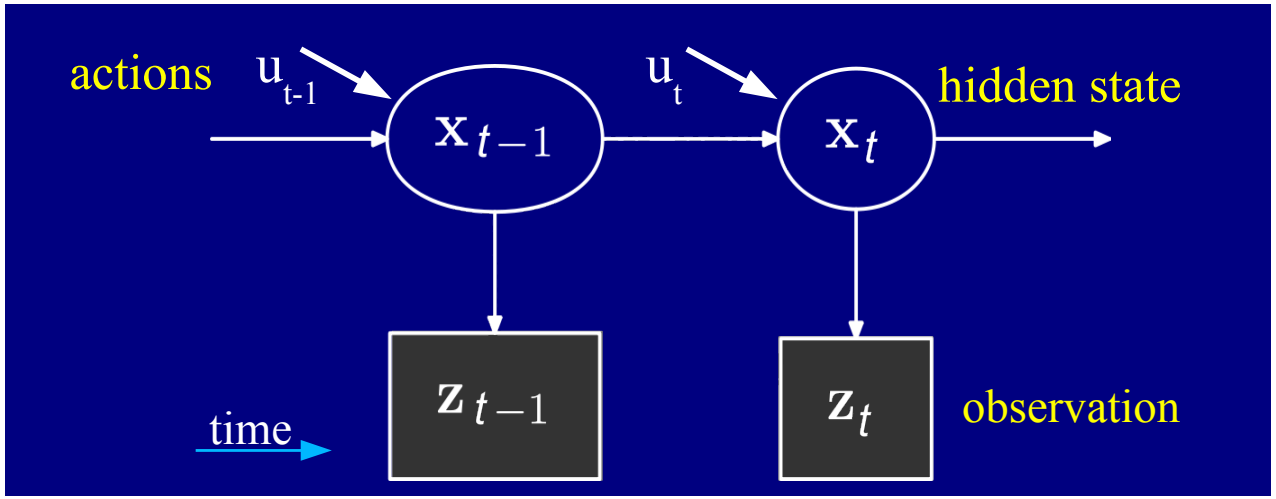
Dynamic Distributional Clauses

Prior distribution $p(x_0)$

State transition model $p(x_t|x_{t-1},u_t)$

Measurement model $p(z_t|x_t)$

Other rules: $p(x'_t|x''_t)$



Magnetic scenario

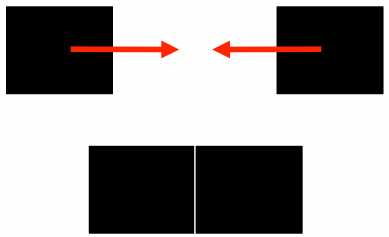
- 3 object types: magnetic, ferromagnetic, nonmagnetic

$$type(X)_t \sim finite([1/3:magnet, 1/3:ferromagnetic, 1/3:nonmagnetic]) \leftarrow object(X).$$

- 2 magnets attract or repulse

$$interaction(A,B)_t \sim finite([0.5:attraction, 0.5:repulsion]) \leftarrow object(A), object(B), A<B, type(A)_t = magnet, type(B)_t = magnet.$$

- Next position after attraction



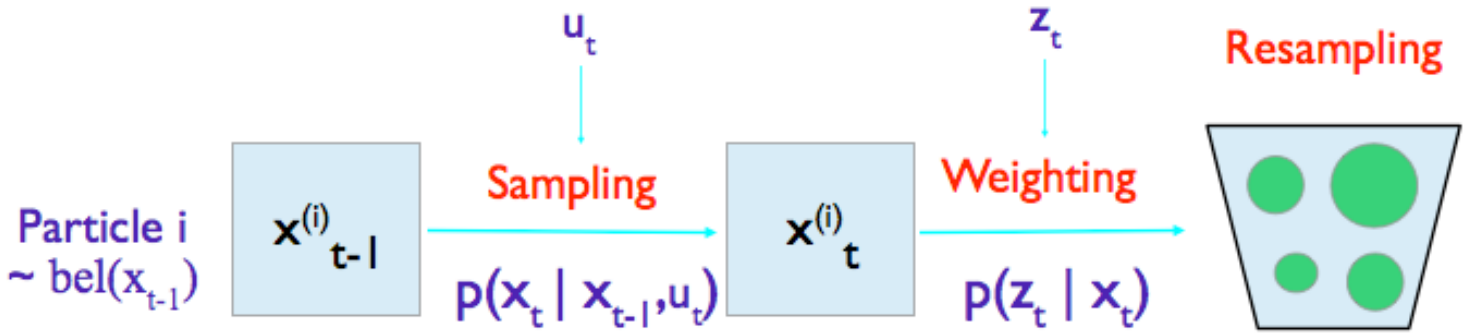
$$pos(A)_{t+1} \sim gaussian(middlepoint(A,B)_t, Cov) \leftarrow near(A,B)_t, not(held(A)), not(held(B)), interaction(A,B)_t = attr, c/dist(A,B)_t^2 > friction(A)_t.$$

$$pos(A)_{t+1} \sim gaussian(pos(A)_t, Cov) \leftarrow not(\ attraction(A,B)\ ).$$

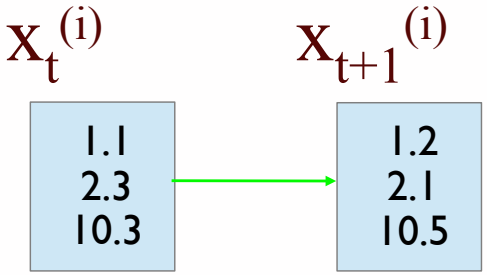
Particle Filter

(Sequential Monte Carlo)

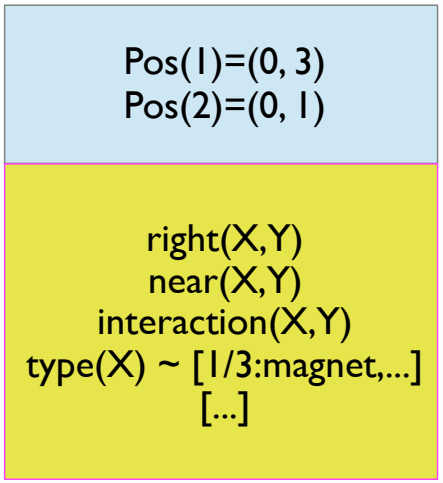
- Based on sampling → approximate inference
- Particles (samples) to represent bel(x_t)



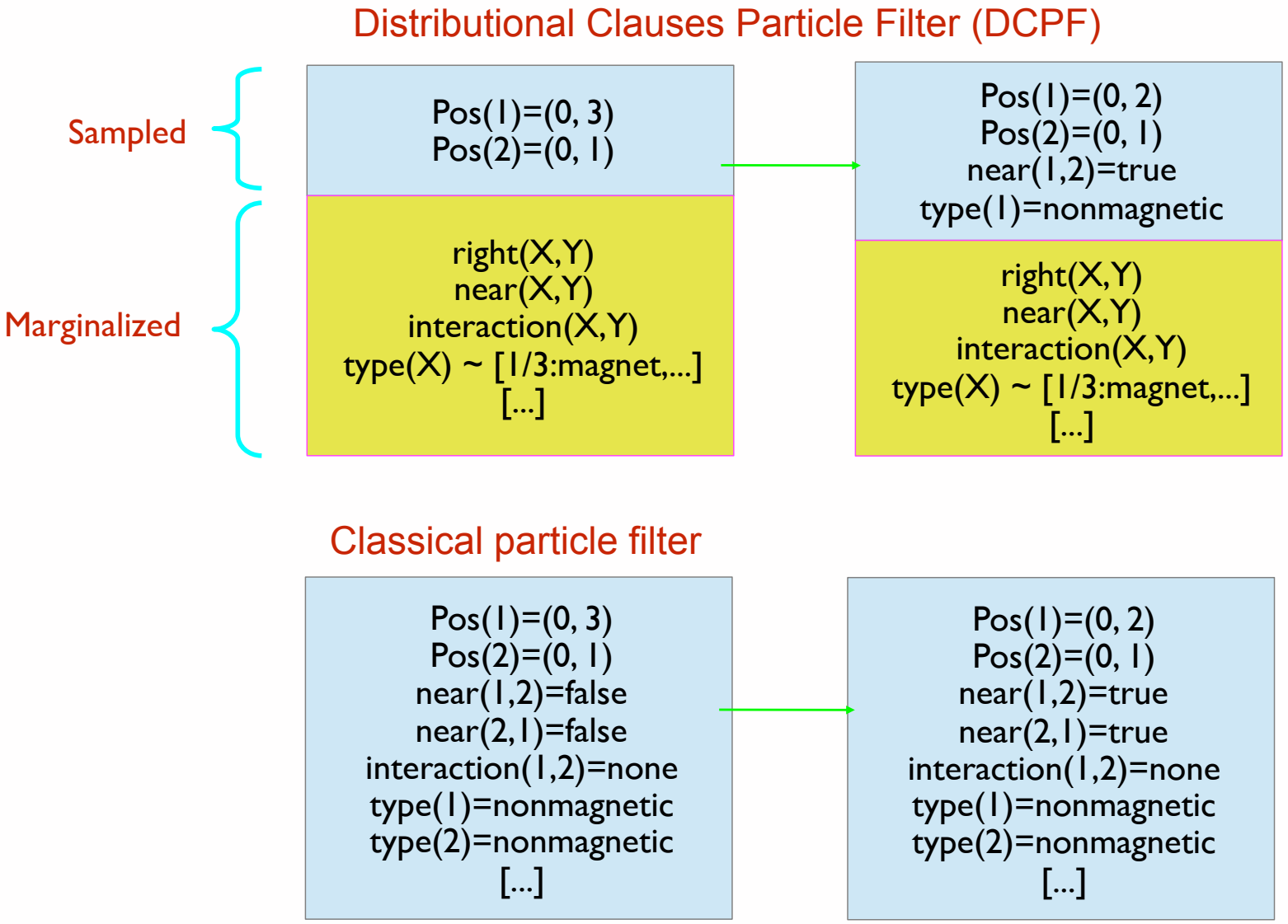
Classical Particle Filter vs DCPF



- Classical PF
  - Fixed set of random variables
  - Update the entire state
- DCPF
  - **Adaptive state** (particle): the number of facts / random variables can change over time
  - Particles are partial interpretations
  - Expressive language

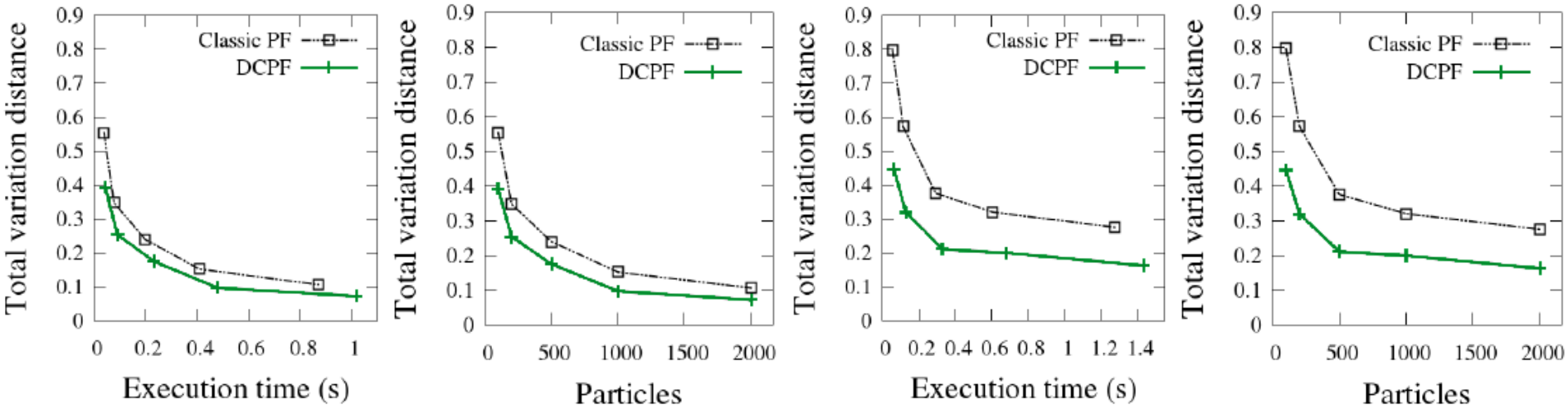


Optimized inference: partial state



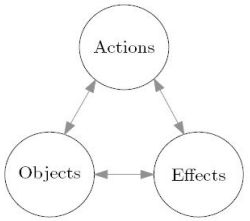
Experiments

- Particles are partial state, remaining variables are marginalized
- Better performance in bigger models



Learning relational affordances

Learn probabilistic model



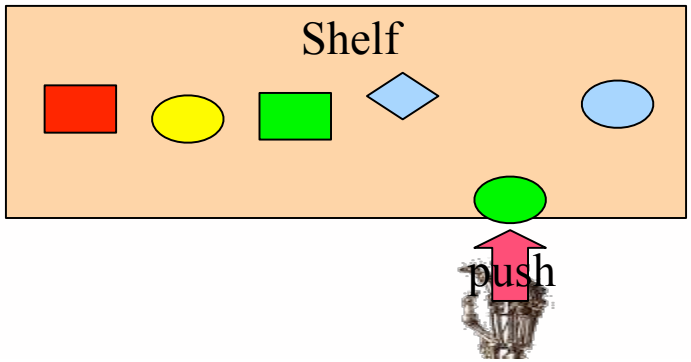
Inputs	Outputs	Function
(O, A)	E	Effect prediction
(O, E)	A	Action recognition/planning
(A, E)	O	Object recognition/selection

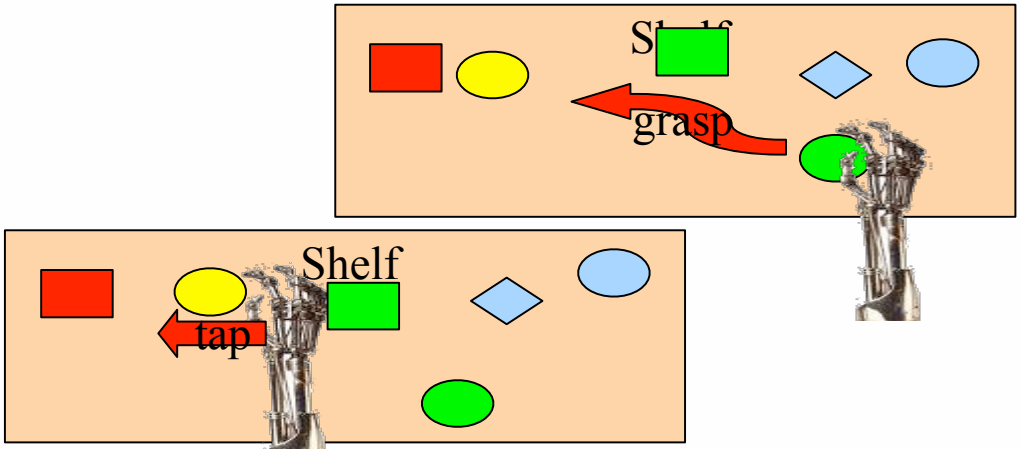
Learning relational affordances between two objects (learnt by experience)

From two object interactions

Generalize to N

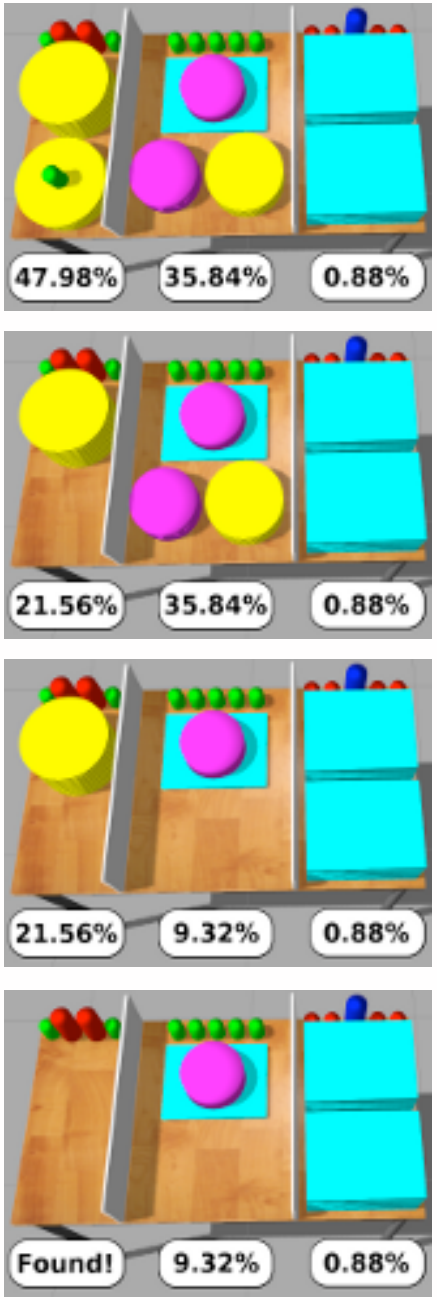
Moldovan et al. ICRA 12, 13, 14





Occluded Object Search

- How to achieve a specific configuration of objects on the shelf?
- Where's the orange mug?
- Where's something to serve soup in?
- Models of objects and their spatial arrangement



[Moldovan *et al. 14*]

ProbLog for activity recognition from video

CAVIAR-INRIA human activity dataset

28 videos
≈ 26.500 frames

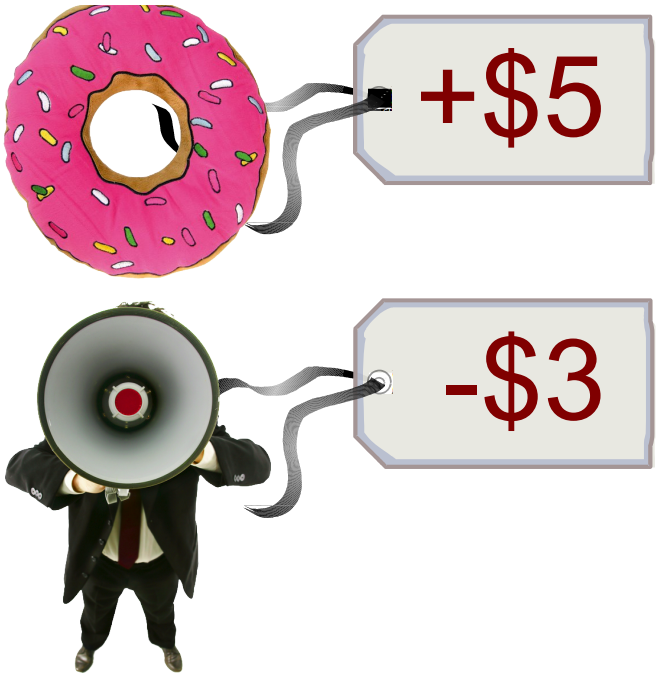
- Separation between low-level events (LLE) and high-level events (HLE)
  - LLE: *walking, running, active, inactive, abrupt*
  - HLE: *meeting, moving, fighting, leaving_object*
- Probabilistic Logic approach: *Event Calculus in ProbLog* (Prob-EC) to infer the high-level events from an **algebra** of low-level events.
- Example:

$$initiatedAt(fighting(P_1, P_2) = true, T) \leftarrow$$
$$happensAt(abrupt(P_1), T),$$
$$holdsAt(close(P_1, P_2, 44) = true, T),$$
$$not\ happensAt(inactive(P_2), T).$$

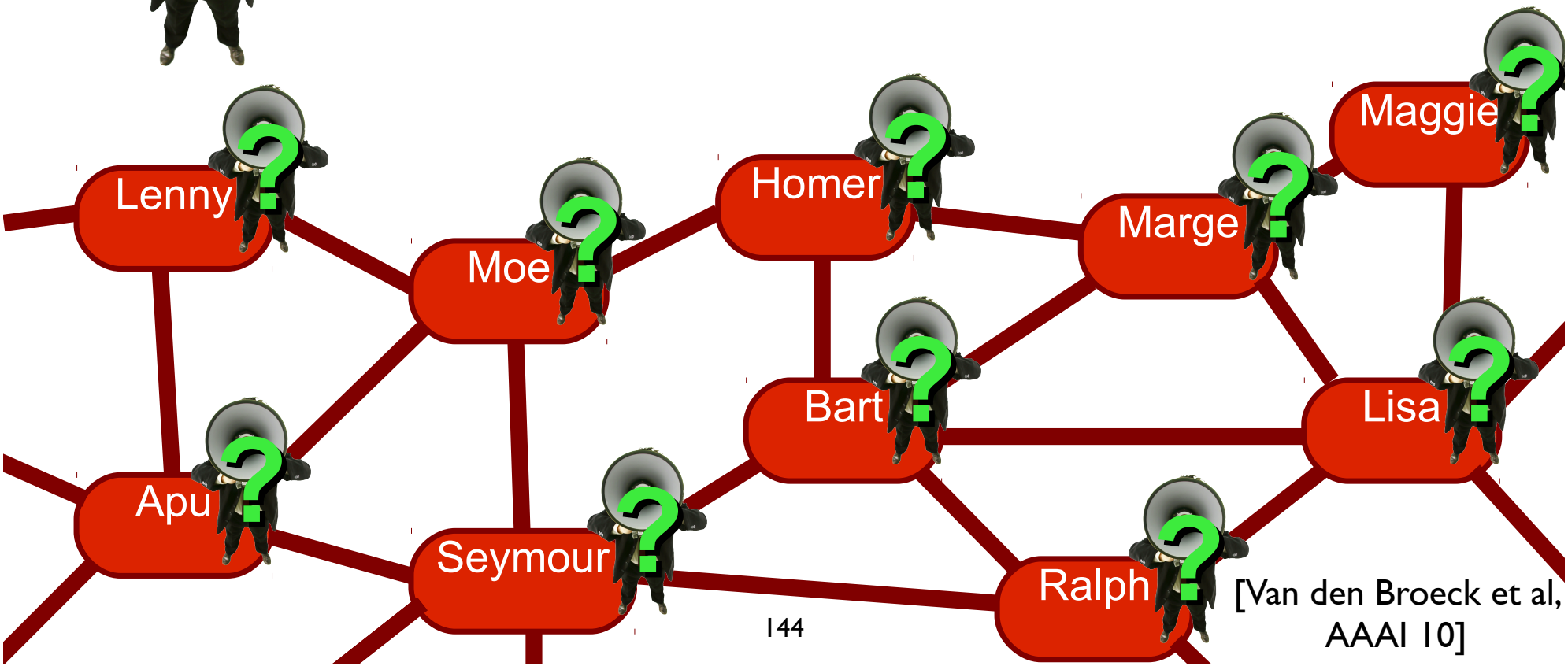
[Skarlatidis et al, TPLP 13]

decisions

Viral Marketing

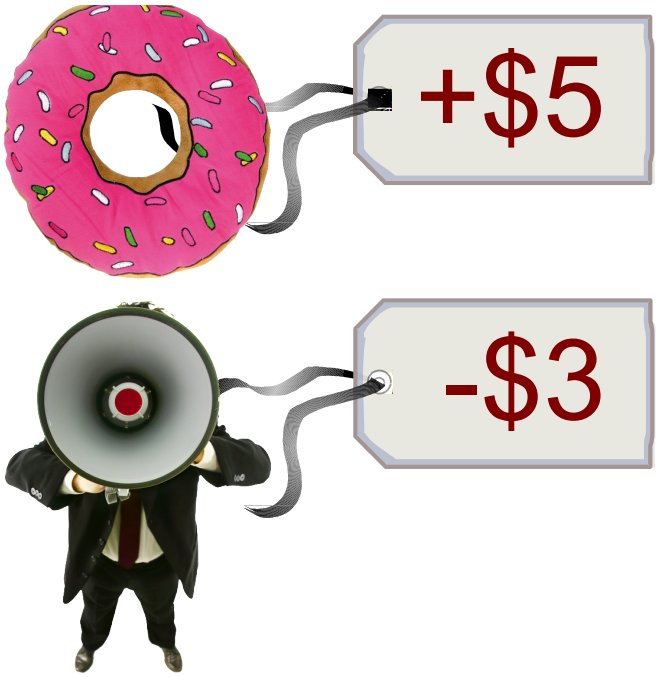


Which advertising strategy maximizes expected profit?

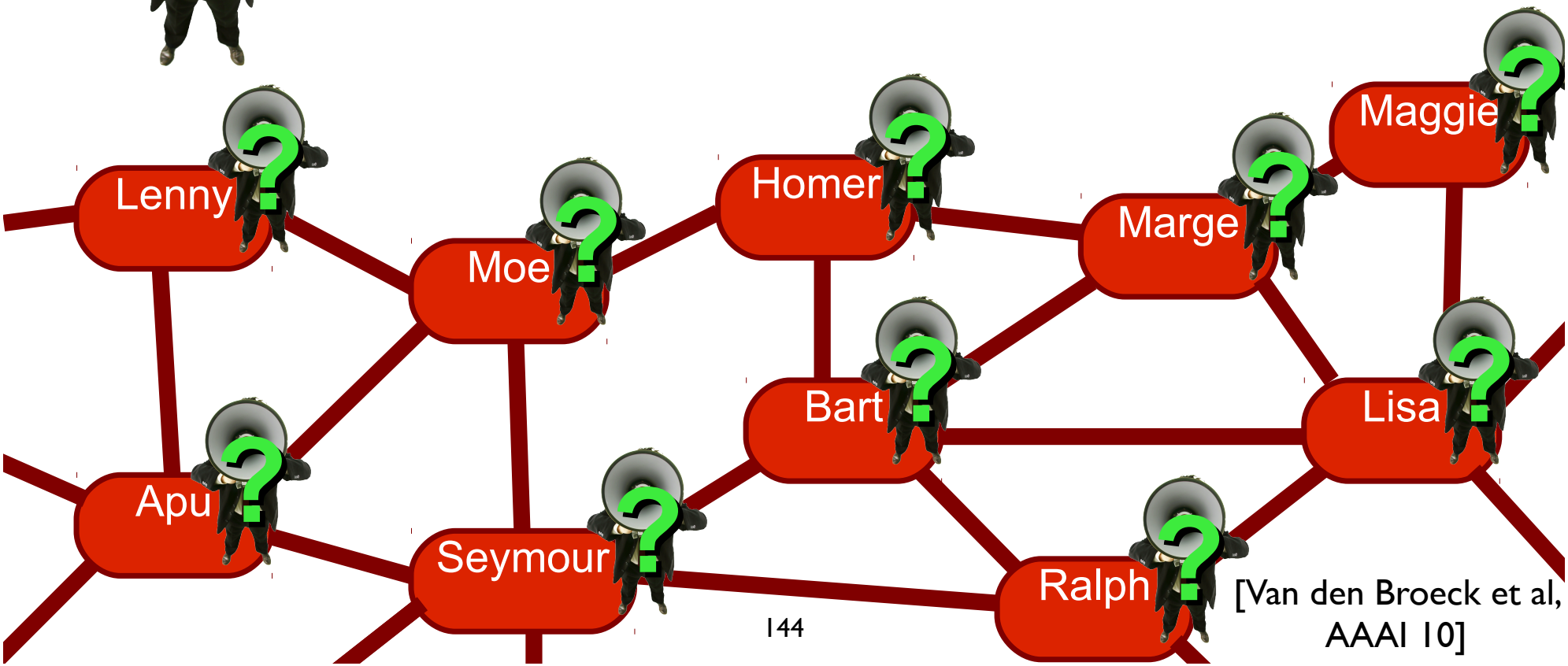


[Van den Broeck et al, AAAI 10]

Viral Marketing

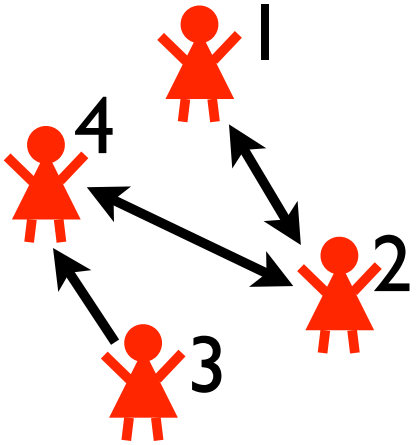


decide truth values of some atoms



[Van den Broeck et al, AAAI 10]

DTProbLog

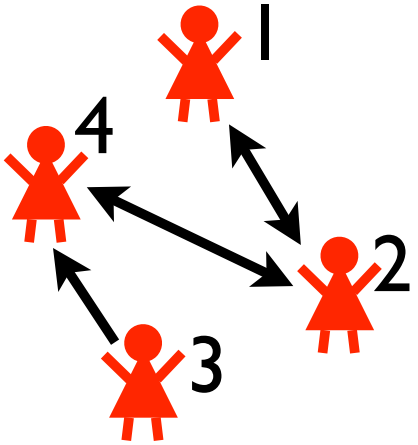



```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog

```
? :: marketed(P) :- person(P).
```

decision fact: true or false?



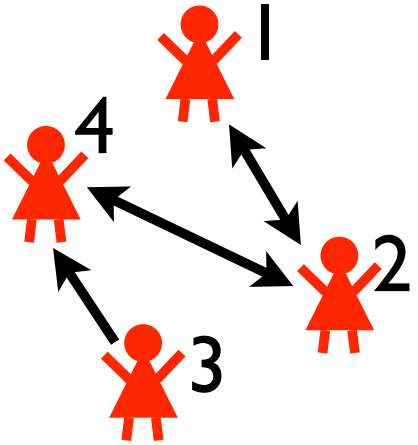

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog

```
? :: marketed(P) :- person(P).
```

```
0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).
```

probabilistic facts
+ logical rules

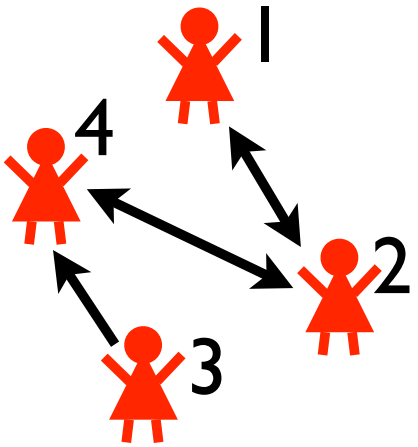



```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog

```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

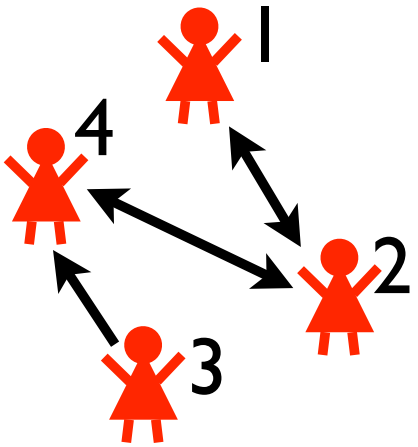
utility facts: cost/reward if true




```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog

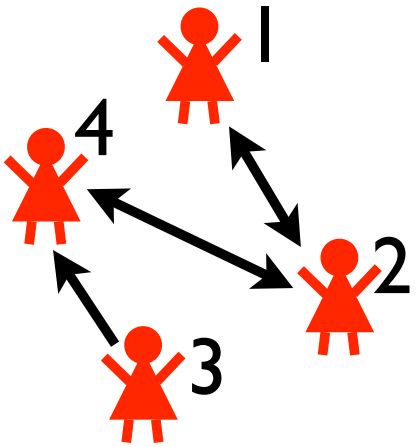
```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```




```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog

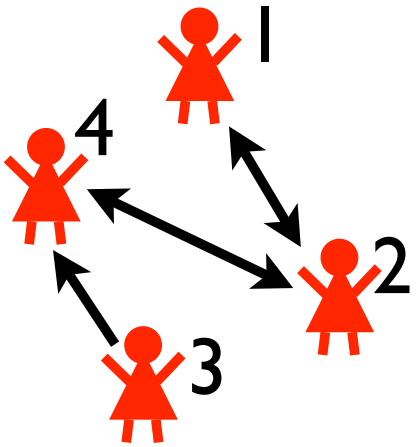
```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```




```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog

```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

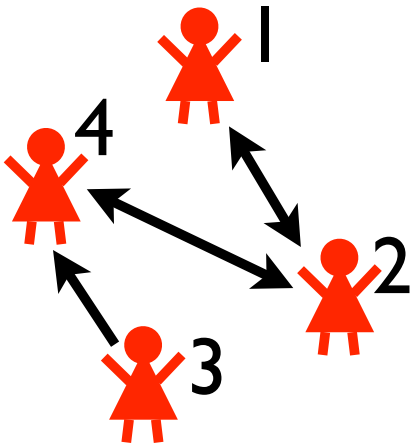



```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

```
marketed(1)        marketed(3)
```

DTProbLog

```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

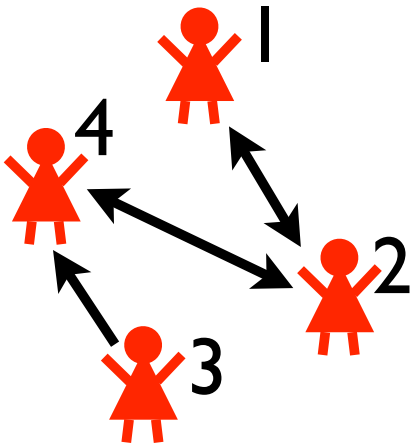



```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

```
marketed(1)        marketed(3)

    bt(2,1)   bt(2,4)        bm(1)
```

DTProbLog

```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```



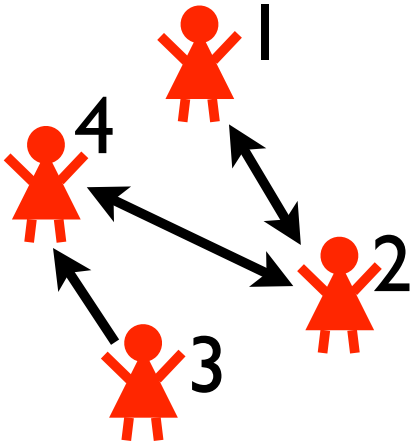

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog

```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

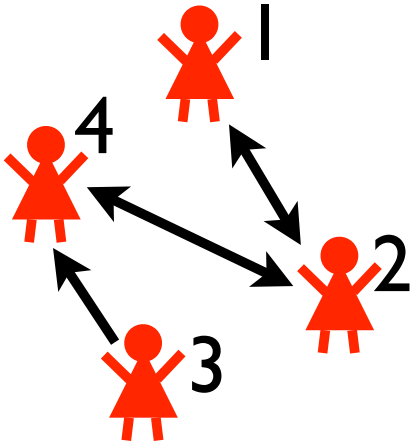
utility = −3 + −3 + 5 + 5 = 4

probability = 0.0032




```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog




```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

utility = −3 + −3 + 5 + 5 = 4

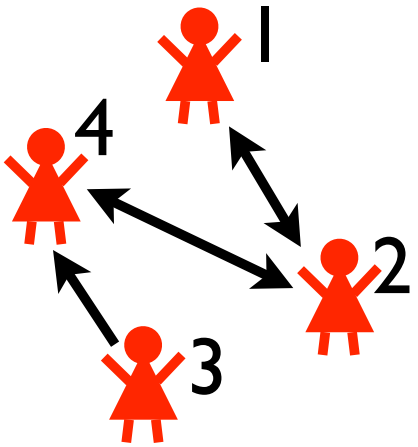
probability = 0.0032

```
marketed(1)        marketed(3)

    bt(2,1)   bt(2,4)        bm(1)

buys(1)     buys(2)
```

world contributes 0.0032×4 to expected utility of strategy

DTProbLog

```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

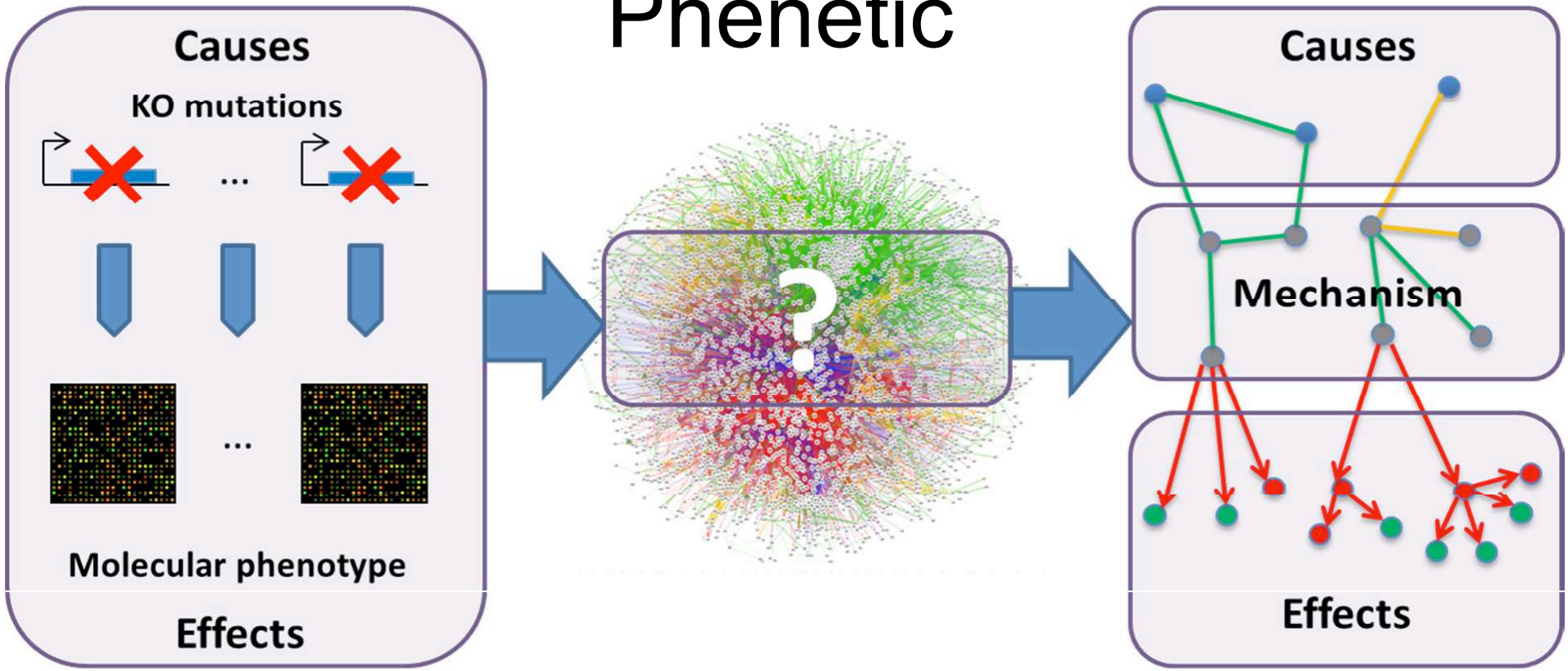



```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

task: find strategy that maximizes expected utility

solution: using ProbLog technology

Phenetic



- Causes: Mutations
  - All related to similar phenotype
- Effects: Differentially expressed genes
- 27 000 cause effect pairs

- Interaction network:
  - 3063 nodes
    - Genes
    - Proteins
  - 16794 edges
    - Molecular interactions
    - Uncertain

- Goal: connect causes to effects through common subnetwork
  - = Find mechanism
- Techniques:
  - DTProbLog
  - Approximate inference

[De Maeyer et al., Molecular Biosystems 13]

Roadmap

- Modeling
- Inference
- Learning & Dynamics
- Advanced Inference and KBMC

... with some detours on the way

Part IV: Advanced Inference and KBMC

Complexity of Querying

Complexity of querying

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

0.96x0.99=0.95

0.9x0.93=0.83

0.87x0.93=0.80

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

Complexity of querying

ProducesProduct

Company	Product	P
...	...	...

HeadquarteredIn

Company	City	P
...	...	...

```
select distinct x.Product, x.Company,
                x.P * y.P as P
from ProducesProduct x, HeadquarteredIn y
where x.Company = y.Company
      and y.City = 'san_jose'
order by P desc
```

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

0.96x0.99=0.95

0.9x0.93=0.83

0.87x0.93=0.80

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

Complexity of querying

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
result(Product,Company) :-
    producesProduct(Company,Product),
    headquarteredIn(Company,san_jose).
query(result(_,_)).
```

Complexity of querying

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
result(Product,Company) :-
    producesProduct(Company,Product),
    headquarteredIn(Company,san_jose).
query(result(_,_)).
```

each ground query has a **single proof**

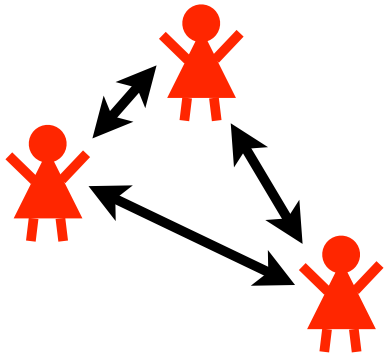
→ no disjoint-sum-problem,

easy evaluation

Complexity of querying in probabilistic databases

- queries have fixed size (no recursion)
- size of query << size of database
- complexity of evaluating given query measured in size of database (data complexity)
- previous example: polynomial
  (as all standard relational database queries)

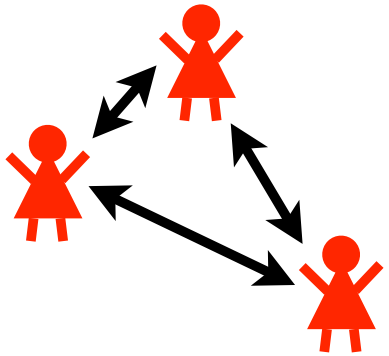
How hard is evaluating q?



```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
              person(X), person(Y), X \= Y.

q :- stress(X), influences(X,Y).
```

```
person(1).
person(2).
person(3).
```

How hard is evaluating q?



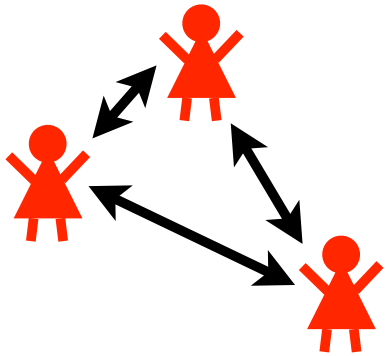
```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
                person(X), person(Y), X \= Y.

q :- stress(X), influences(X,Y).
```

```
person(1).
person(2).
person(3).
```

proofs

```
s(1),i(1,2)
s(1),i(1,3)
s(2),i(2,1)
s(2),i(2,3)
s(3),i(3,1)
s(3),i(3,2)
```

How hard is evaluating q?



```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
                person(X), person(Y), X \= Y.

q :- stress(X), influences(X,Y).
```

```
person(1).
person(2).
person(3).
```

proofs

s(1),i(1,2)

s(1),i(1,3)

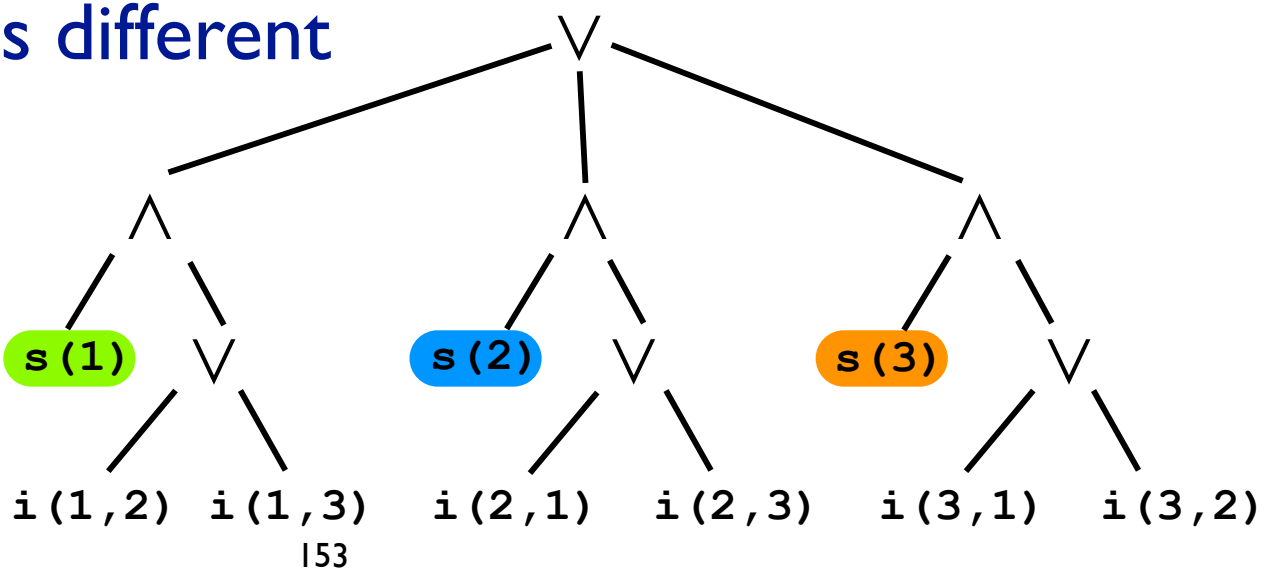
s(2),i(2,1)

s(2),i(2,3)

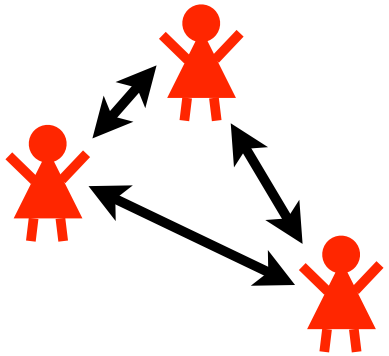
s(3),i(3,1)

s(3),i(3,2)

tree structure, all leaves different



How hard is evaluating q?



```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
                person(X), person(Y), X \= Y.

q :- stress(X), influences(X,Y).
```

```
person(1).
person(2).
person(3).
```

proofs

s(1),i(1,2)

s(1),i(1,3)

s(2),i(2,1)

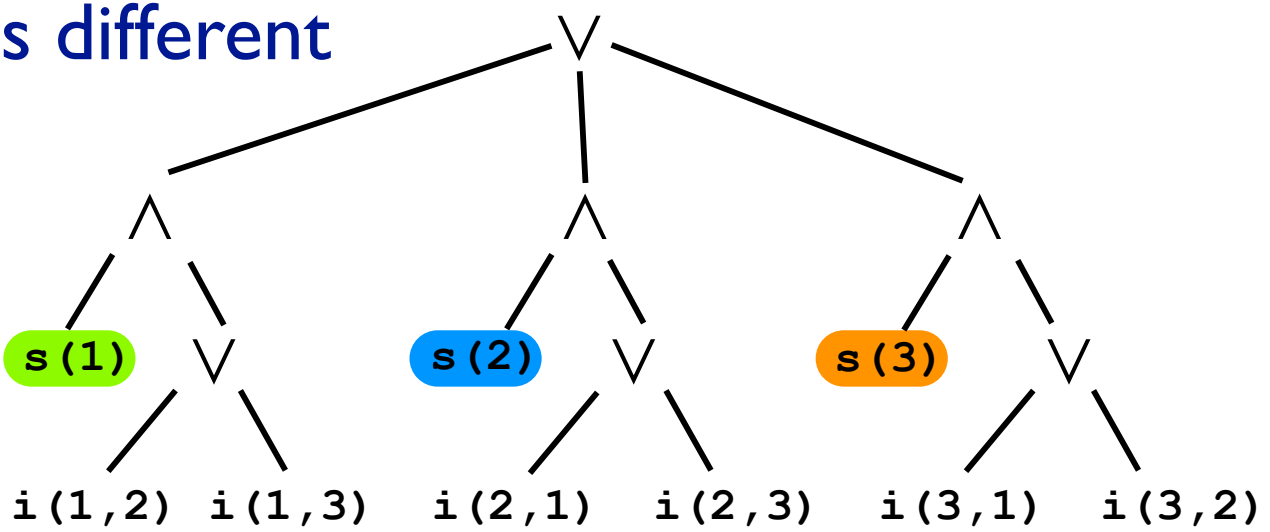
s(2),i(2,3)

s(3),i(3,1)

s(3),i(3,2)

tree structure, all leaves different

$$P(\varphi_1 \wedge \varphi_2) = P(\varphi_1) \cdot P(\varphi_2)$$
$$P(\varphi_1 \vee \varphi_2) = 1 - (1 - P(\varphi_1)) \cdot (1 - P(\varphi_2))$$



$$P(\mathtt{q}) = 1 - \prod_{j=1..n} (1 - \left(P(\mathtt{s}(\mathtt{X_j}))(1 - \prod_{k=1..n, k \neq j} (1 - P(\mathtt{i}(\mathtt{X_j}, \mathtt{Y_k})))) \right))$$

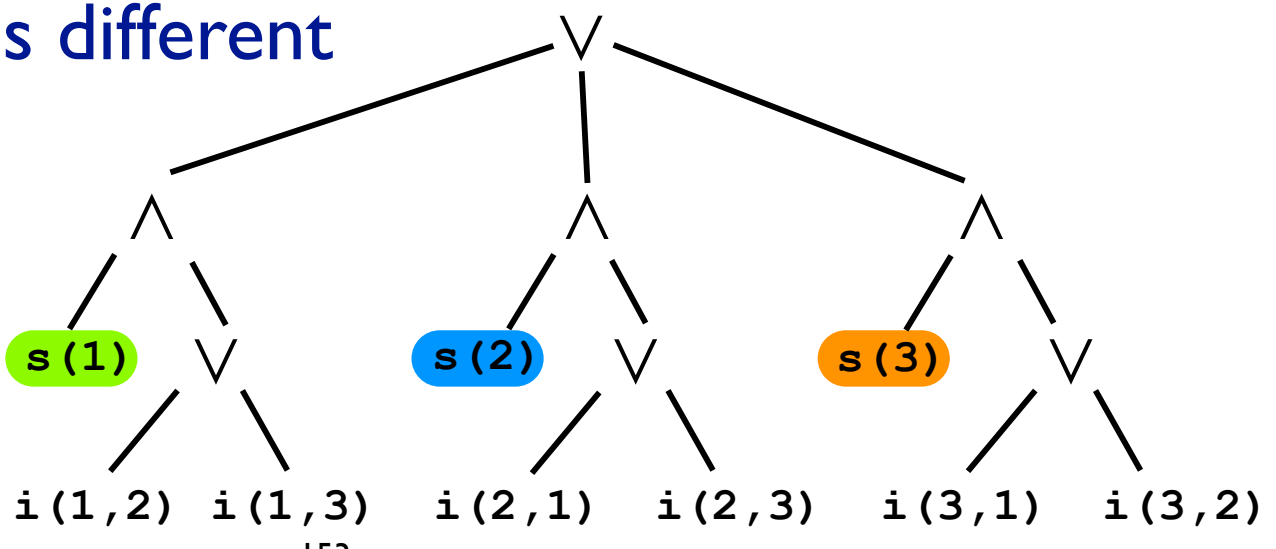
polynomial in database size / number of persons n

proofs

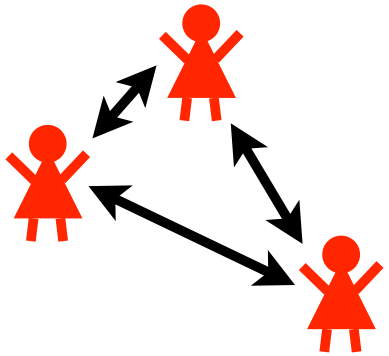
s(1),i(1,2)
s(1),i(1,3)
s(2),i(2,1)
s(2),i(2,3)
s(3),i(3,1)
s(3),i(3,2)

tree structure, all leaves different

$$P(\varphi_1 \wedge \varphi_2) = P(\varphi_1) \cdot P(\varphi_2)$$
$$P(\varphi_1 \vee \varphi_2) = 1 - (1 - P(\varphi_1)) \cdot (1 - P(\varphi_2))$$



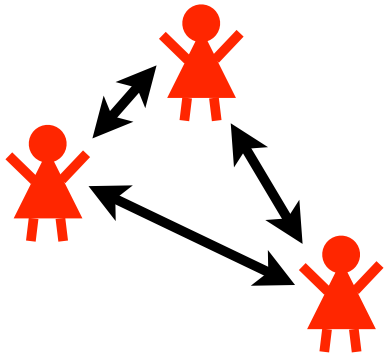
How hard is evaluating q?



```
0.3::stress(X):- person(X).
0.5::male(X)  :- person(X).
0.2::influences(X,Y):-
                person(X), person(Y), X \= Y.

q :- stress(X), influences(X,Y), male(Y).
```

```
person(1).
person(2).
person(3).
```

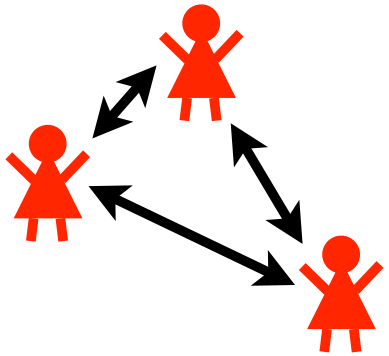
How hard is evaluating q?



```
0.3::stress(X):- person(X).
0.5::male(X) :- person(X).
0.2::influences(X,Y):-
              person(X), person(Y), X \= Y.

q :- stress(X), influences(X,Y), male(Y).
```

```
person(1).
person(2).
person(3).
```

How hard is evaluating q?



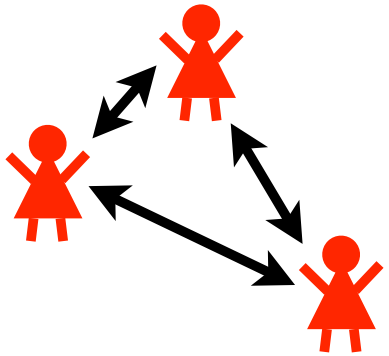
```
0.3::stress(X):- person(X).
0.5::male(X) :- person(X).
0.2::influences(X,Y):-
                person(X), person(Y), X \= Y.

q :- stress(X), influences(X,Y), male(Y).
```

```
person(1).
person(2).
person(3).
```

proofs

```
s(1),i(1,2),m(2)
s(1),i(1,3),m(3)
s(2),i(2,1),m(1)
s(2),i(2,3),m(3)
s(3),i(3,1),m(1)
s(3),i(3,2),m(2)
```

How hard is evaluating q?



```
0.3::stress(X):- person(X).
0.5::male(X)  :- person(X).
0.2::influences(X,Y):-
                person(X), person(Y), X \= Y.

q :- stress(X), influences(X,Y), male(Y).
```

```
person(1).
person(2).
person(3).
```

proofs

s(1),i(1,2),m(2)

s(1),i(1,3),m(3)

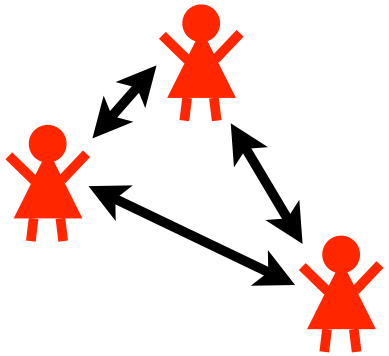
s(2),i(2,1),m(1)

s(2),i(2,3),m(3)

s(3),i(3,1),m(1)

s(3),i(3,2),m(2)

How hard is evaluating q?

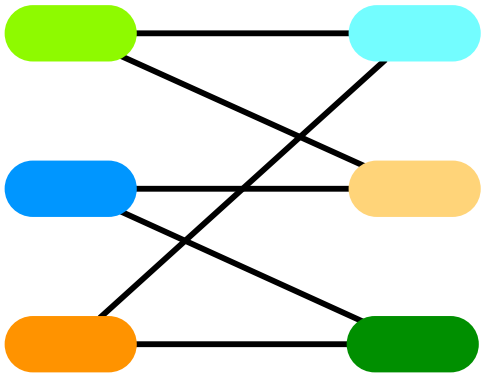


```
0.3::stress(X):- person(X).
0.5::male(X)  :- person(X).
0.2::influences(X,Y):-
                person(X), person(Y), X \= Y.

q :- stress(X), influences(X,Y), male(Y).
```

```
person(1).
person(2).
person(3).
```

proofs

```
s(1),i(1,2),m(2)
s(1),i(1,3),m(3)
s(2),i(2,1),m(1)
s(2),i(2,3),m(3)
s(3),i(3,1),m(1)
s(3),i(3,2),m(2)
```



cannot build tree structure with all leaves different

Read-Once Formulas

- Propositional formulas that can be rewritten such that each variable occurs at most once (= as tree with all leaves different)

Read-Once Formulas

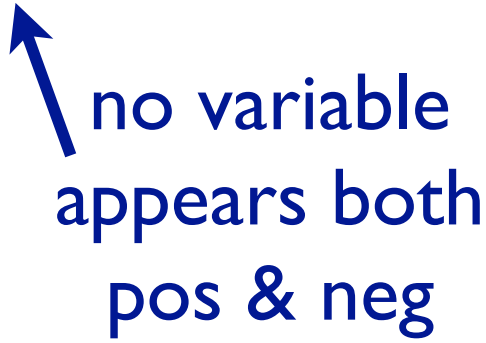
- Propositional formulas that can be rewritten such that each variable occurs at most once (= as tree with all leaves different)
- Can be evaluated in polynomial time

Read-Once Formulas

- Propositional formulas that can be rewritten such that each variable occurs at most once (= as tree with all leaves different)
- Can be evaluated in polynomial time
- Unate formula: read once ↔ P4-free & normal

Read-Once Formulas

- Propositional formulas that can be rewritten such that each variable occurs at most once (= as tree with all leaves different)
- Can be evaluated in polynomial time
- Unate formula: read once ↔ P4-free & normal



$X \vee Y$ is unate

$(X \vee Y) \wedge (\neg X \vee Z)$ not

P4-free and normal

- represent DNF formula as graph:
  - a node for each variable
  - edge (X,Y) $\leftrightarrow$ X,Y in same conjunct

P4-free and normal

- represent DNF formula as graph:
  - a node for each variable
  - edge (X,Y) ↔ X,Y in same conjunct
- **normal**: each clique is part of a conjunct

P4-free and normal

- represent DNF formula as graph:
  - a node for each variable
  - edge (X,Y) ↔ X,Y in same conjunct
- **normal**: each clique is part of a conjunct
- **P4-free**: no induced 4 node subgraph is a path

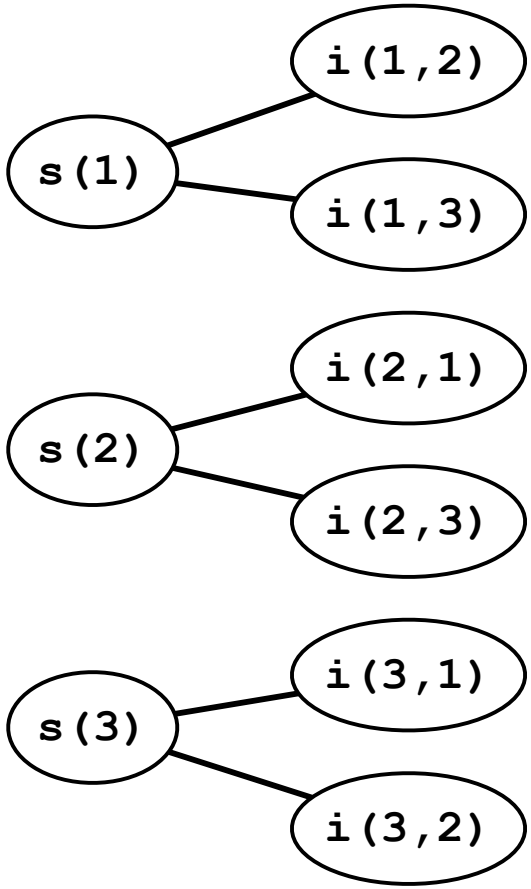
Our first example

```
s(1),i(1,2)
s(1),i(1,3)
s(2),i(2,1)
s(2),i(2,3)
s(3),i(3,1)
s(3),i(3,2)
```

- **normal**: each clique is part of a conjunct
- **P4-free**: no induced 4 node subgraph is a path

Our first example

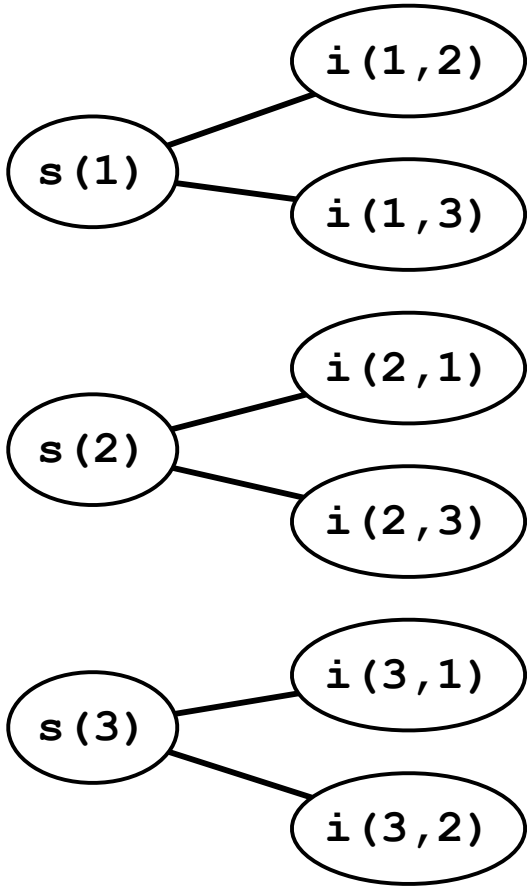
```
s(1),i(1,2)
s(1),i(1,3)
s(2),i(2,1)
s(2),i(2,3)
s(3),i(3,1)
s(3),i(3,2)
```



- **normal**: each clique is part of a conjunct
- **P4-free**: no induced 4 node subgraph is a path

Our first example

```
s(1),i(1,2)
s(1),i(1,3)
s(2),i(2,1)
s(2),i(2,3)
s(3),i(3,1)
s(3),i(3,2)
```



- **normal**: each clique is part of a conjunct ✔
- **P4-free**: no induced 4 node subgraph is a path

Our first example

s(1),i(1,2)

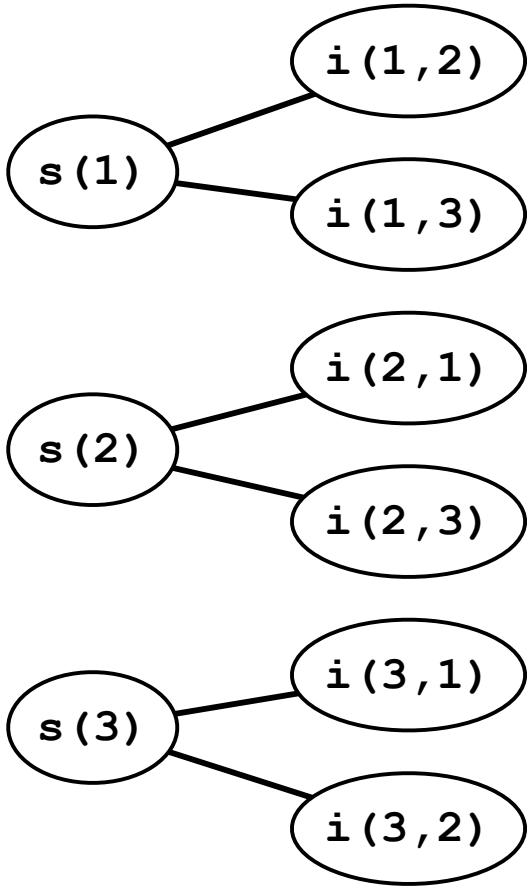
s(1),i(1,3)

s(2),i(2,1)

s(2),i(2,3)

s(3),i(3,1)

s(3),i(3,2)



- **normal**: each clique is part of a conjunct ✔
- **P4-free**: no induced 4 node subgraph is a path ✔

read-once

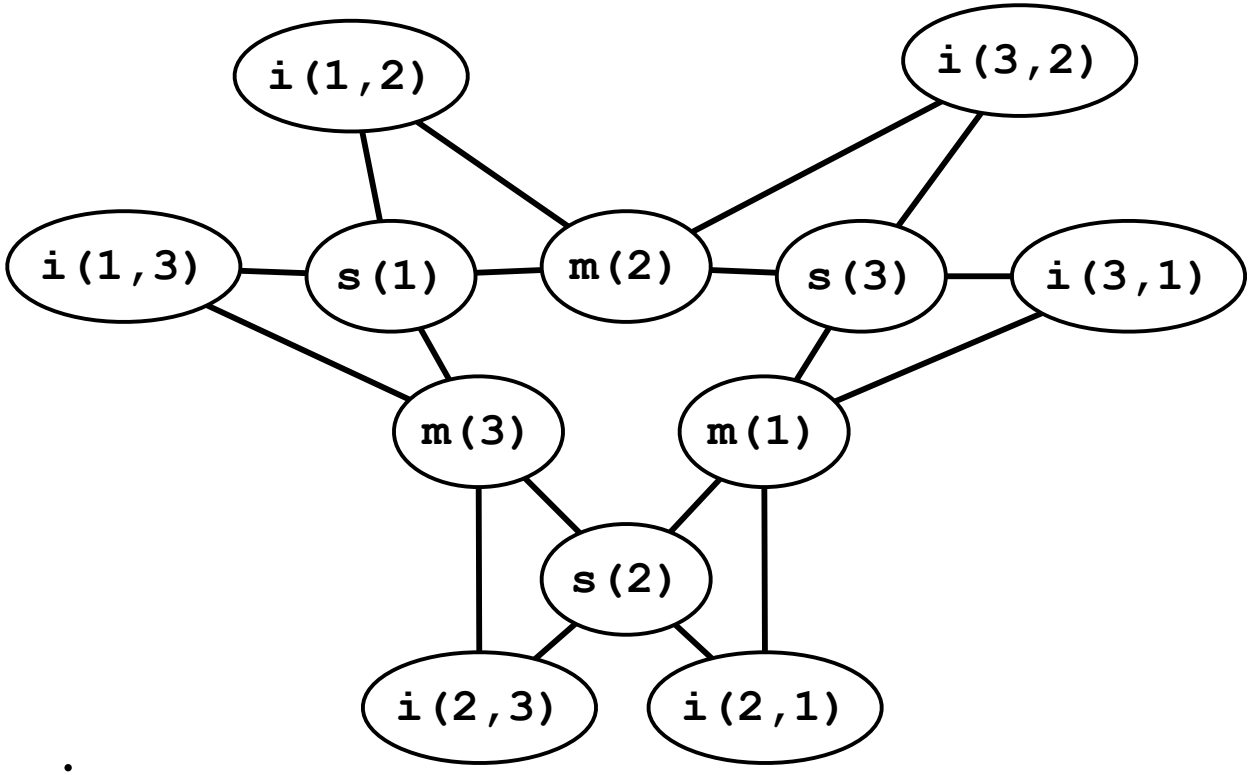
Our second example

```
s(1),i(1,2),m(2)
s(1),i(1,3),m(3)
s(2),i(2,1),m(1)
s(2),i(2,3),m(3)
s(3),i(3,1),m(1)
s(3),i(3,2),m(2)
```

- **normal**: each clique is part of a conjunct
- **P4-free**: no induced 4 node subgraph is a path

Our second example

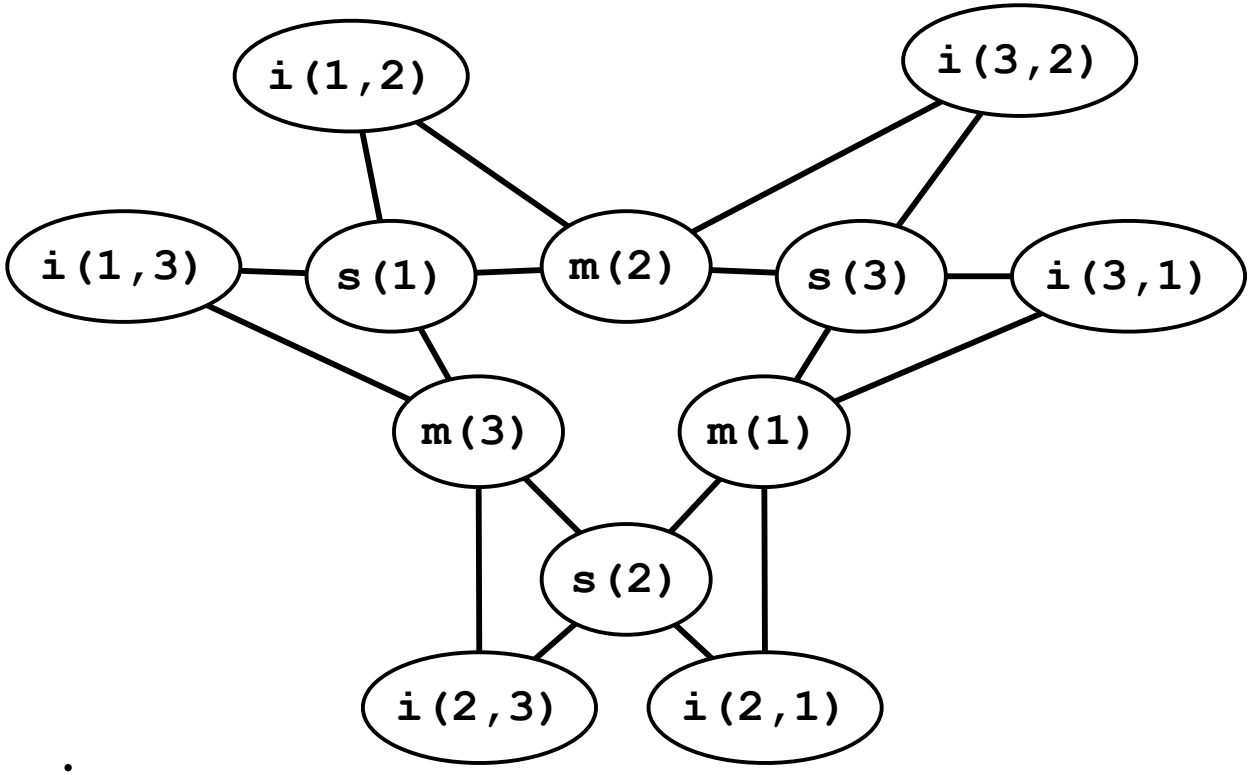
```
s(1),i(1,2),m(2)
s(1),i(1,3),m(3)
s(2),i(2,1),m(1)
s(2),i(2,3),m(3)
s(3),i(3,1),m(1)
s(3),i(3,2),m(2)
```



- **normal**: each clique is part of a conjunct
- **P4-free**: no induced 4 node subgraph is a path

Our second example

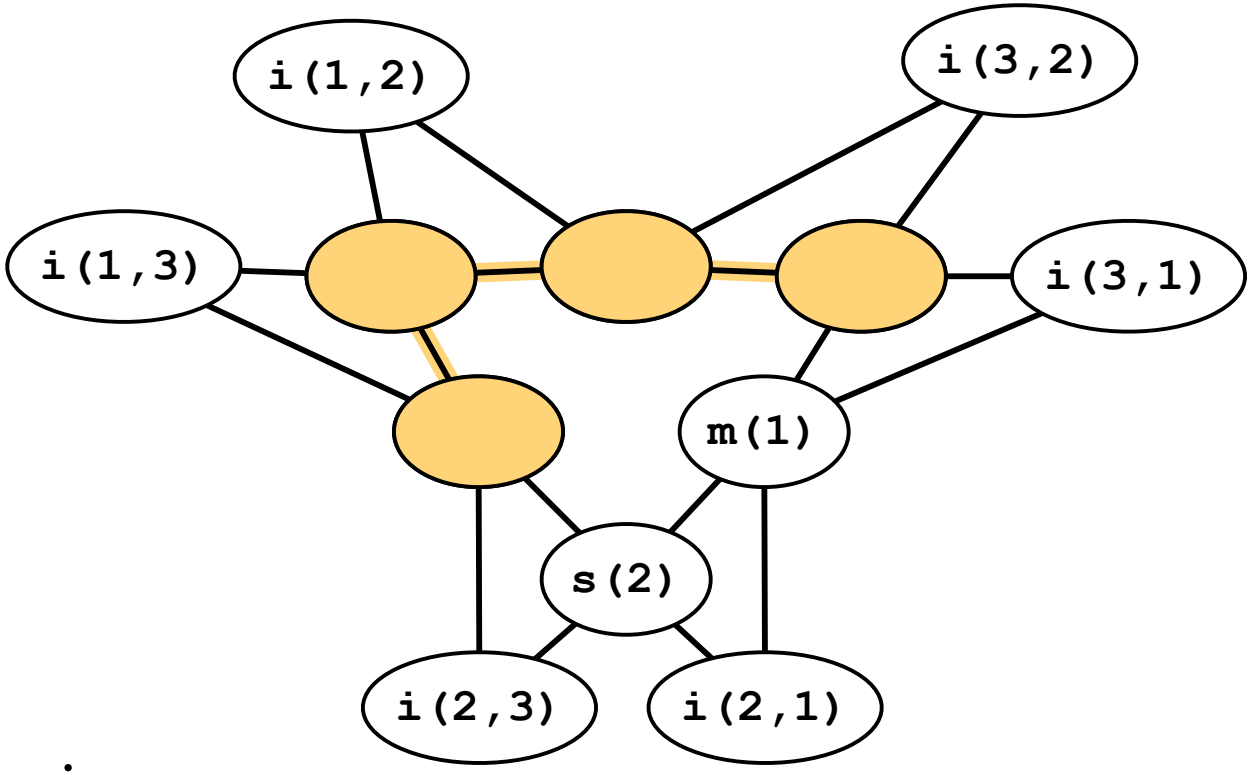
```
s(1),i(1,2),m(2)
s(1),i(1,3),m(3)
s(2),i(2,1),m(1)
s(2),i(2,3),m(3)
s(3),i(3,1),m(1)
s(3),i(3,2),m(2)
```



- **normal**: each clique is part of a conjunct ✔
- **P4-free**: no induced 4 node subgraph is a path

Our second example

```
s(1),i(1,2),m(2)
s(1),i(1,3),m(3)
s(2),i(2,1),m(1)
s(2),i(2,3),m(3)
s(3),i(3,1),m(1)
s(3),i(3,2),m(2)
```



- **normal**: each clique is part of a conjunct ✓
- **P4-free**: no induced 4 node subgraph is a path ✗

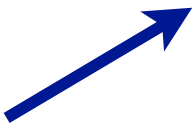
not read-once
(and in fact known to be #P-hard, cf. PDB-book)

Dichotomy of UCQ Evaluation

- **U**nion of **C**onjunctive **Q**ueries
  ≈ Datalog without recursion and negation
- Theorem: UCQ evaluation is either polynomial in database size or #P-hard

Dichotomy of UCQ Evaluation

- **U**nion of **C**onjunctive **Q**ueries
  ≈ Datalog without recursion and negation
- Theorem: UCQ evaluation is either polynomial in database size or #P-hard



counting version of NP decision problems, e.g., model counting

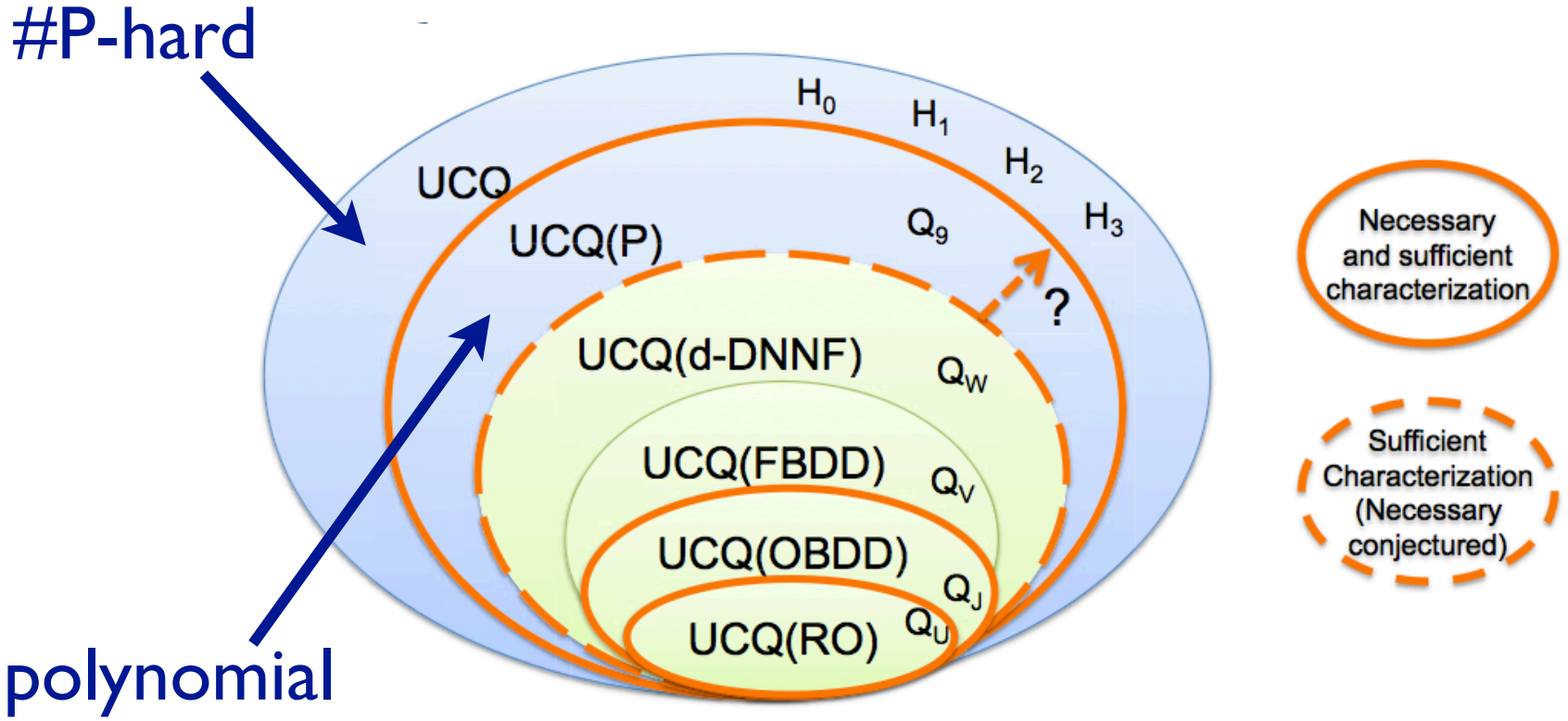


Figure 5.4: The query compilation hierarchy for Unions of Conjunctive Queries (UCQ).

Fig. from [Suciu et al 2011]

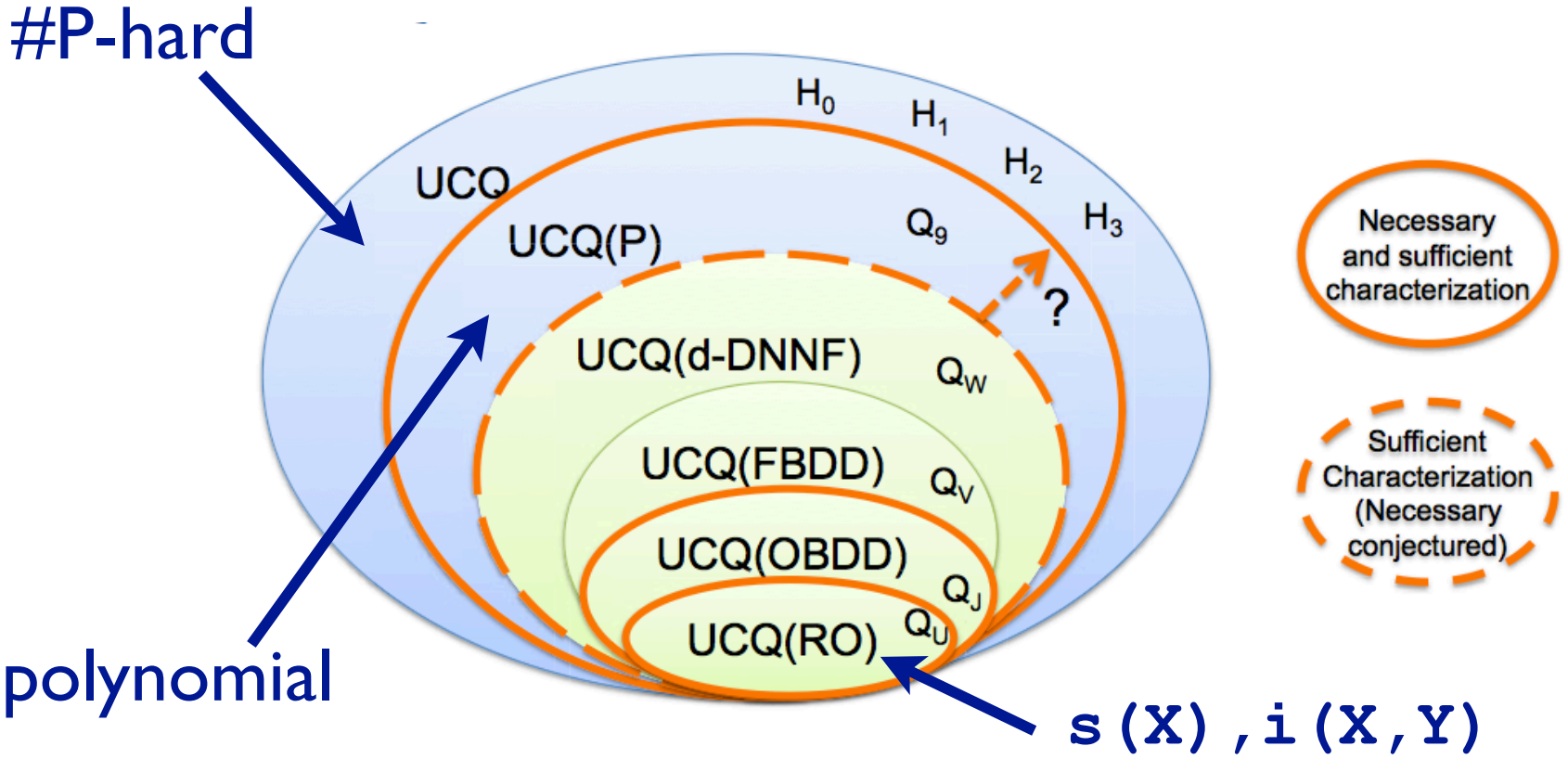


Figure 5.4: The query compilation hierarchy for Unions of Conjunctive Queries (UCQ).

Fig. from [Suciu et al 2011]

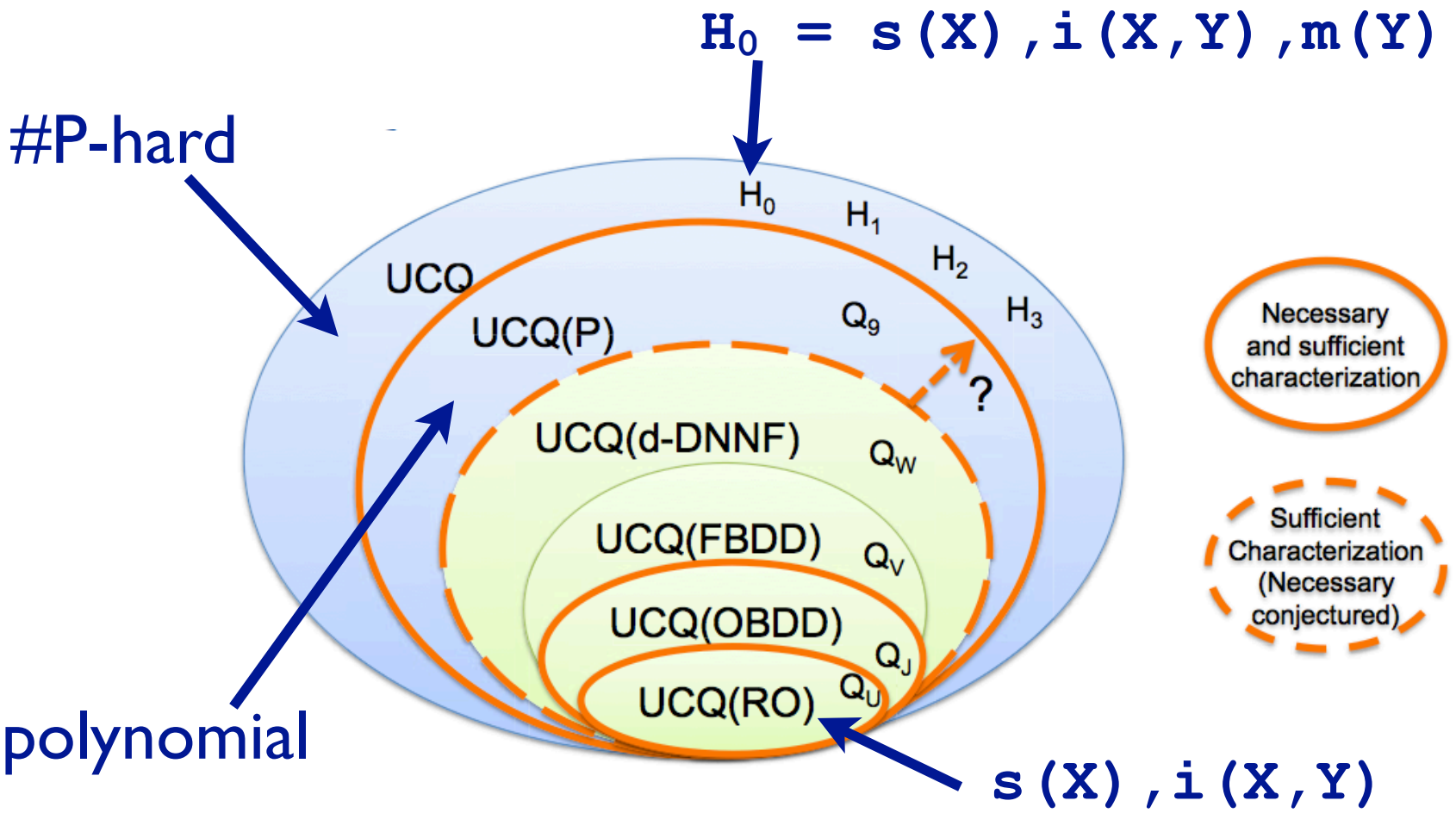


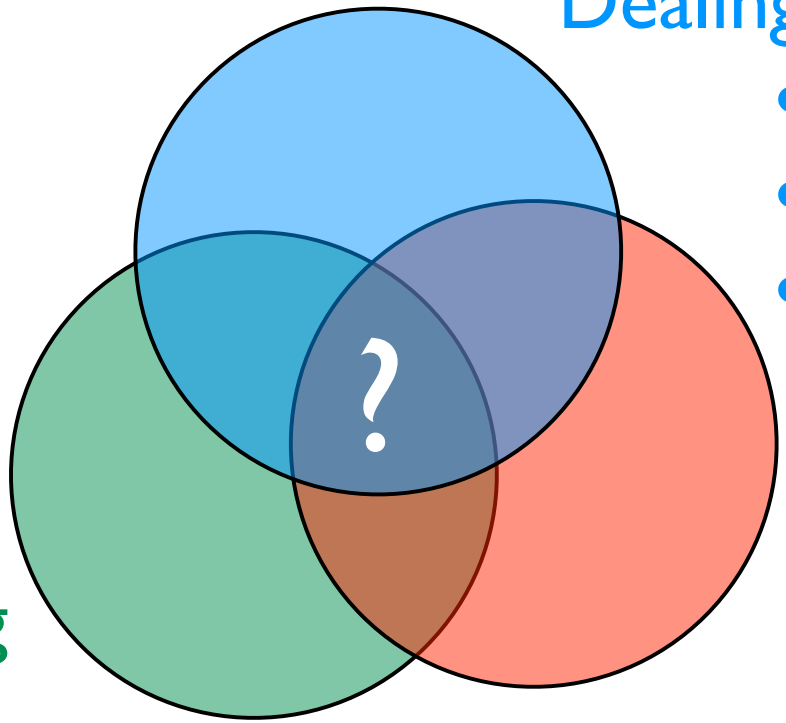
Figure 5.4: The query compilation hierarchy for Unions of Conjunctive Queries (UCQ).

Fig. from [Suciu et al 2011]

A key question in AI:

Dealing with uncertainty
- probability theory
- graphical models
- ...

Reasoning with relational data
- logic
- databases
- programming
- ...



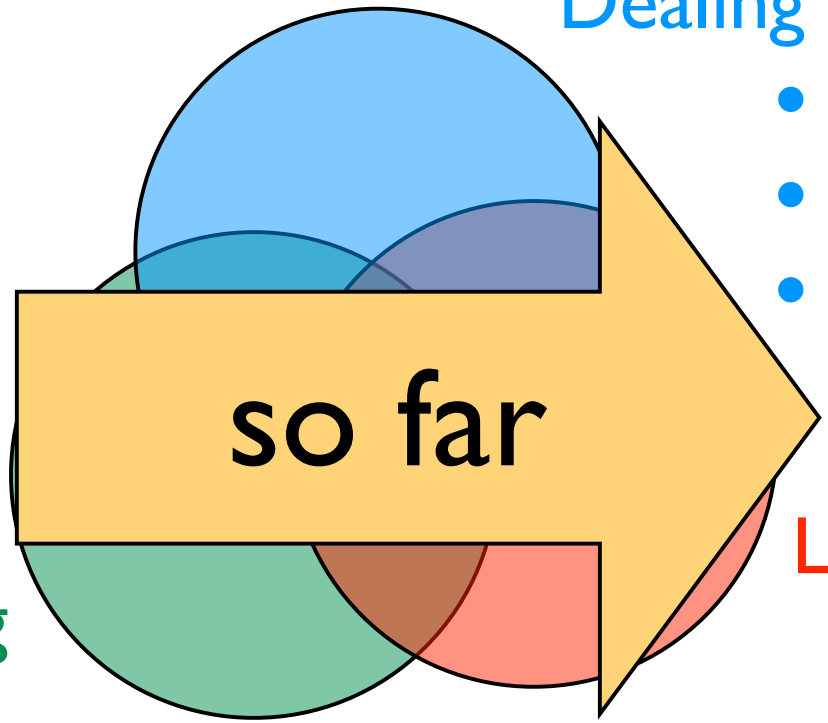
Learning
- parameters
- structure

Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

A key question in AI:

Dealing with uncertainty
- probability theory
- graphical models
- ...

Reasoning with relational data
- logic
- databases
- programming
- ...



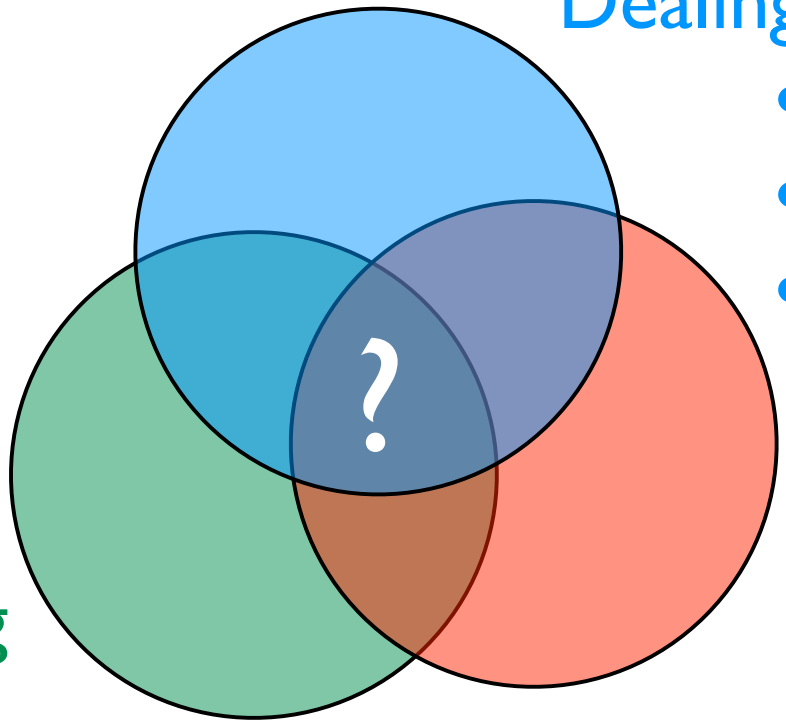
Learning
- parameters
- structure

Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

A key question in AI:

Dealing with uncertainty
- probability theory
- graphical models
- ...

Reasoning with relational data
- logic
- databases
- programming
- ...



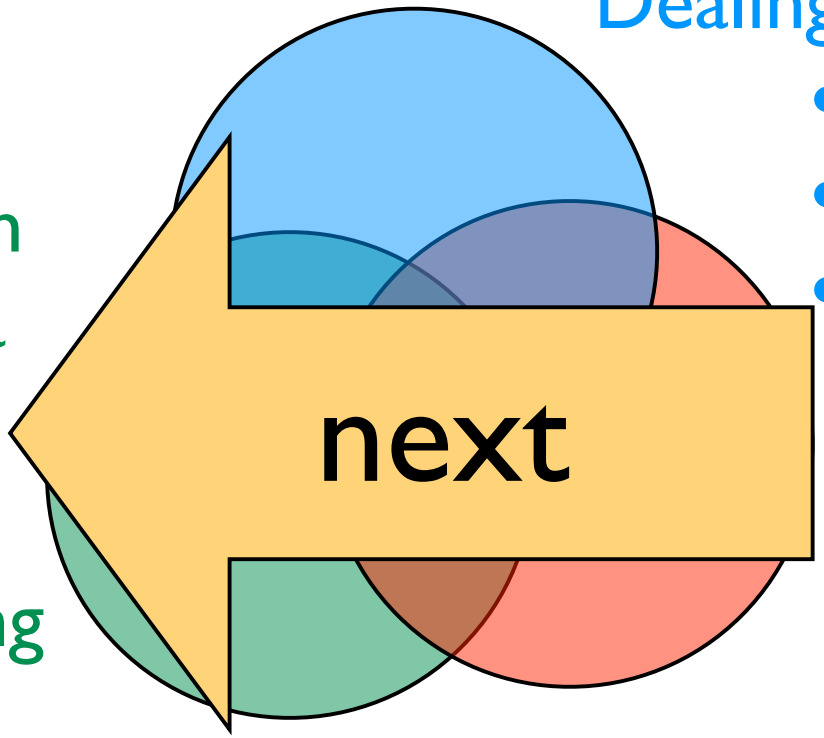
Learning
- parameters
- structure

Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

A key question in AI:

Dealing with uncertainty
- probability theory
- graphical models
- ...

Reasoning with relational data
- logic
- databases
- programming
- ...

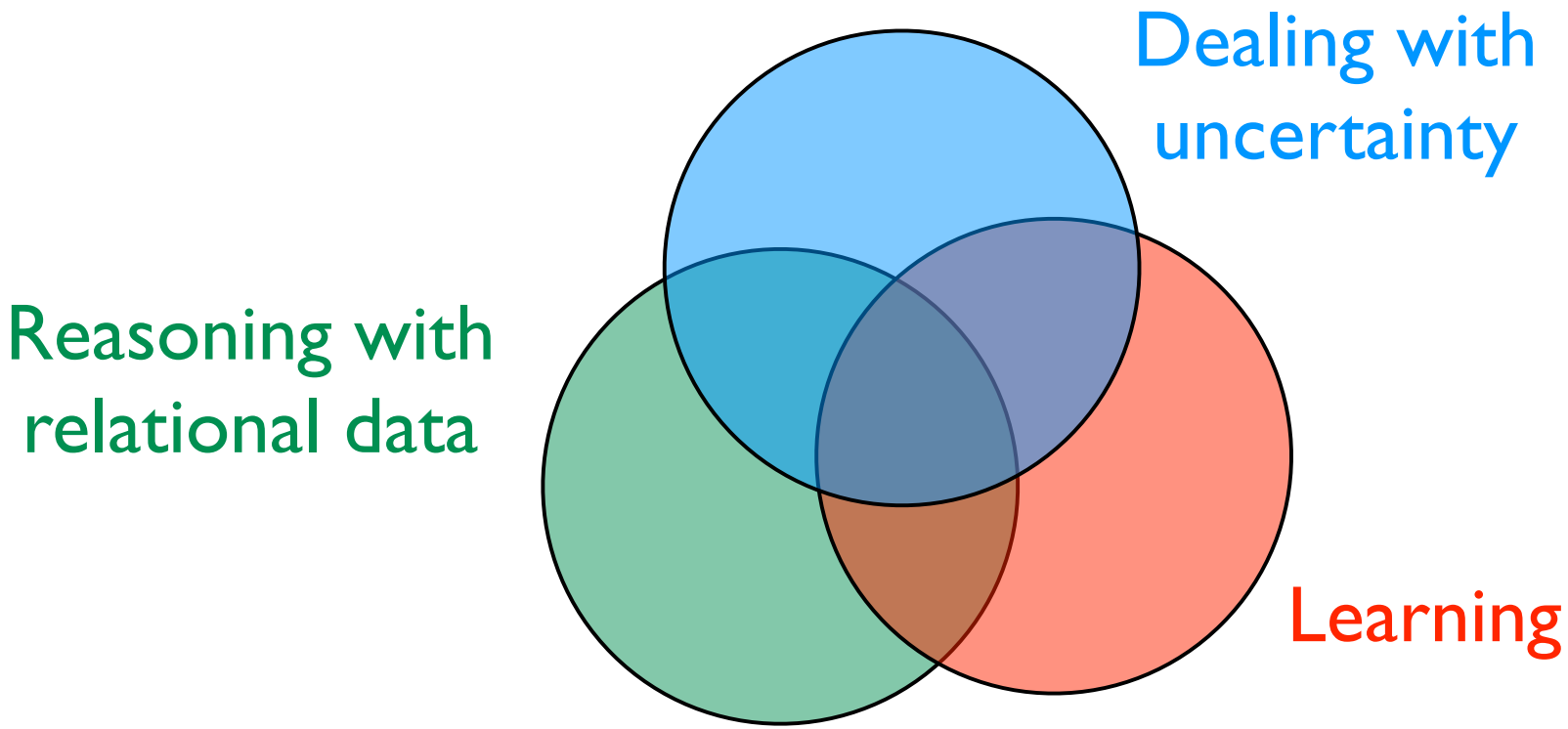


Learning
- parameters
- structure

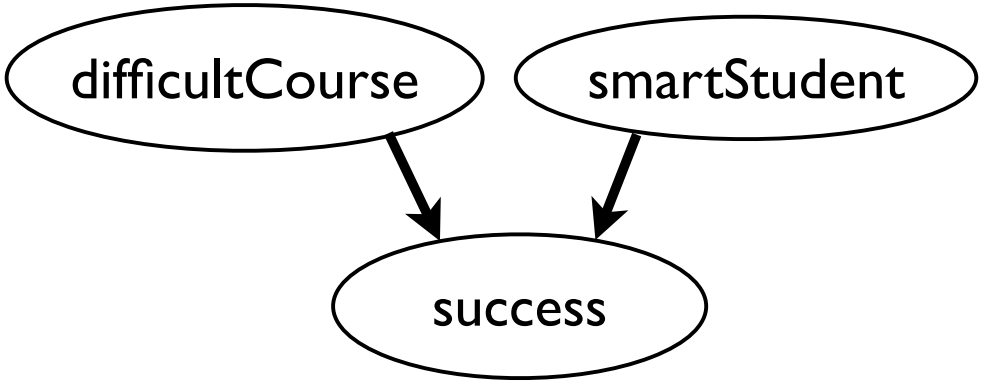
Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

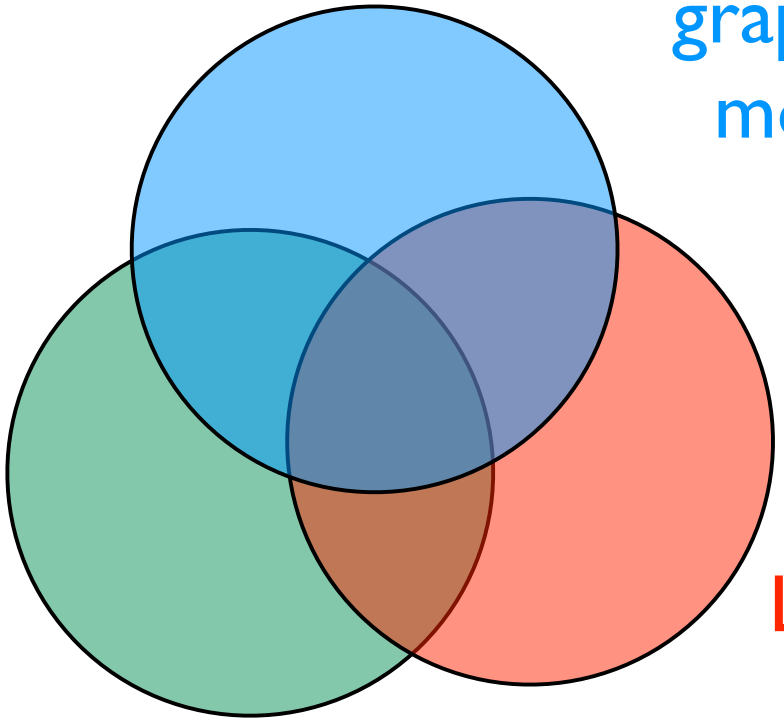
lifted graphical models

Lifted graphical models



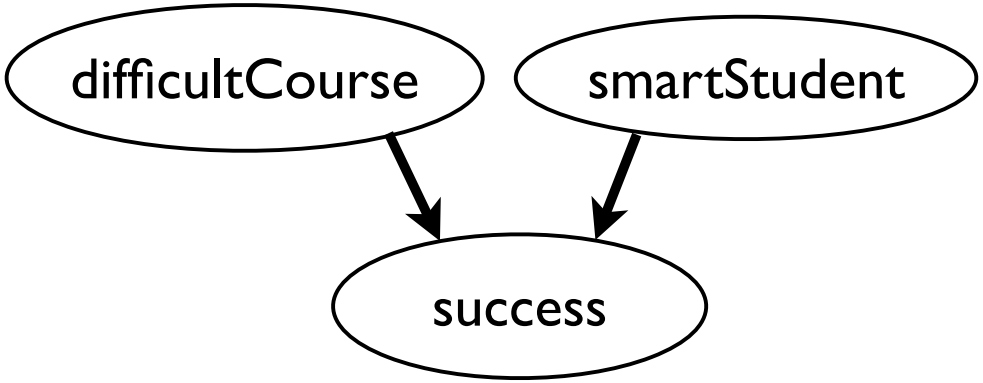
Lifted graphical models





Lifted graphical models

fixed set of random variables



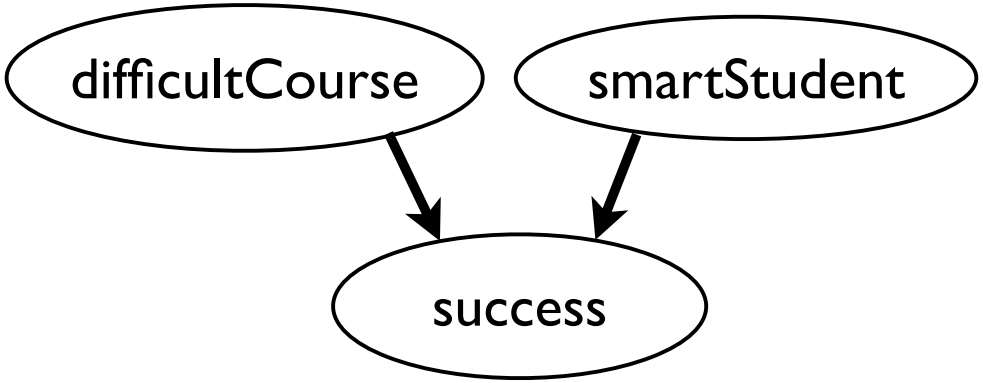
graphical model

Reasoning with relational data

Learning

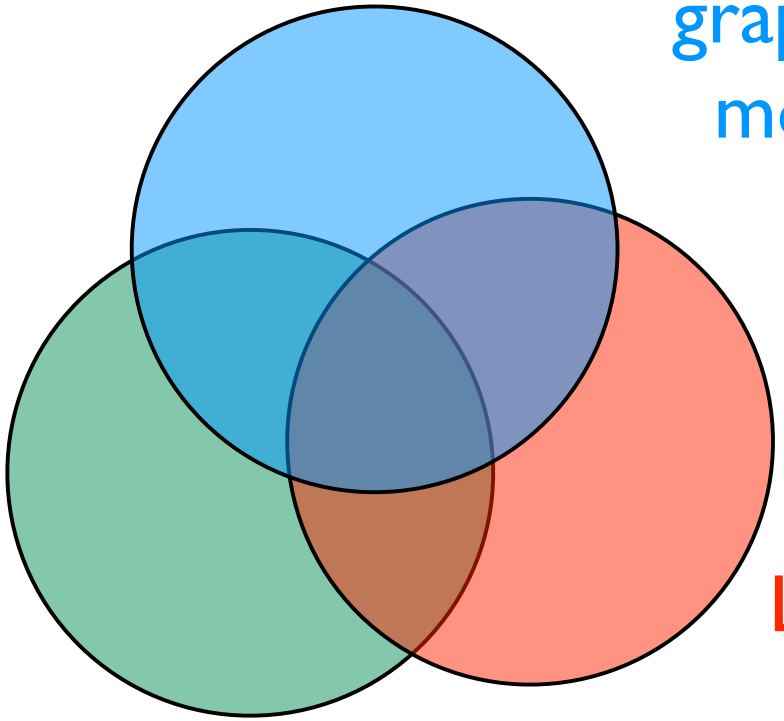
Lifted graphical models

fixed set of random variables

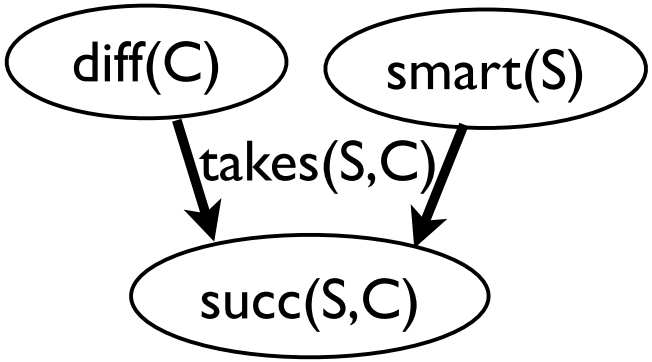


graphical model

relational definition of graphical model

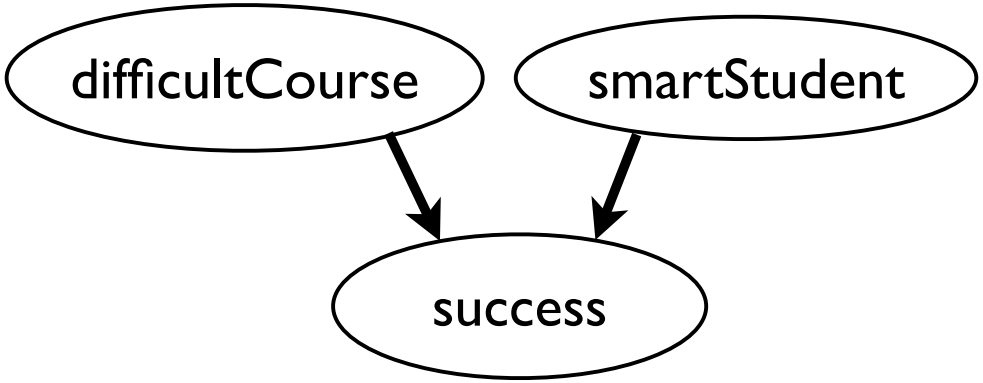


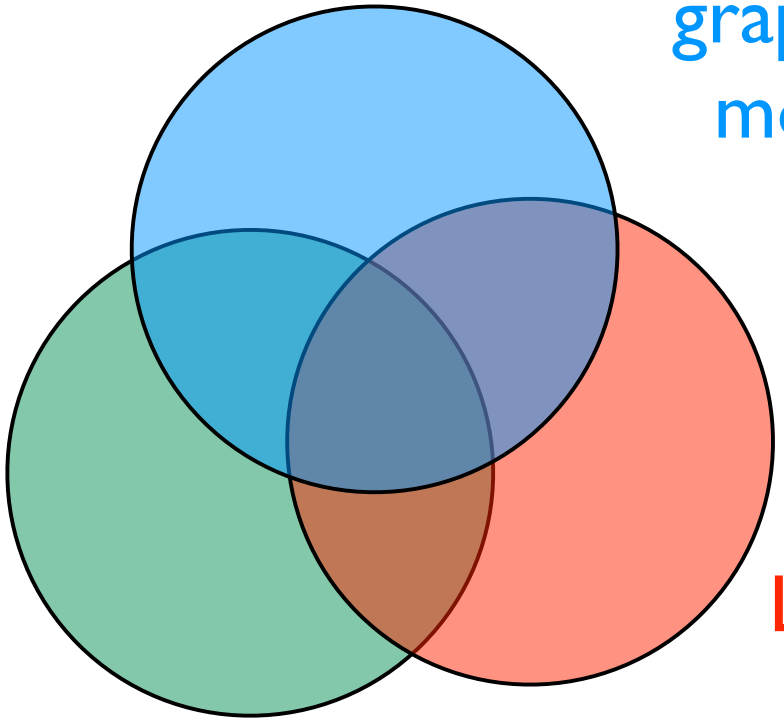
Learning



Lifted graphical models

fixed set of random variables

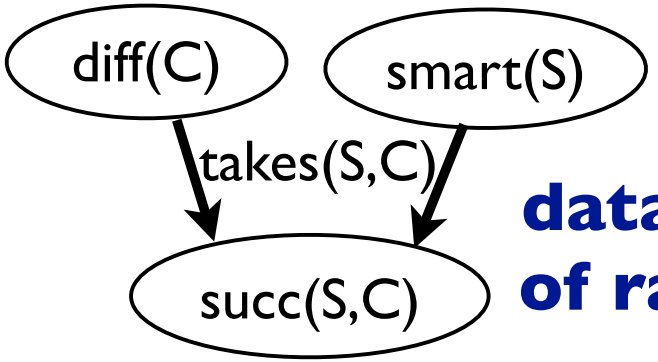




graphical model

relational definition of graphical model

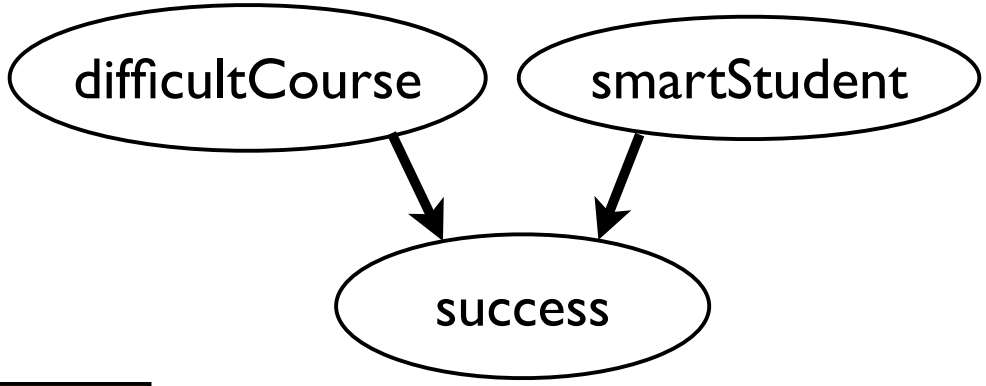
Learning



data-dependent set of random variables

Lifted graphical models

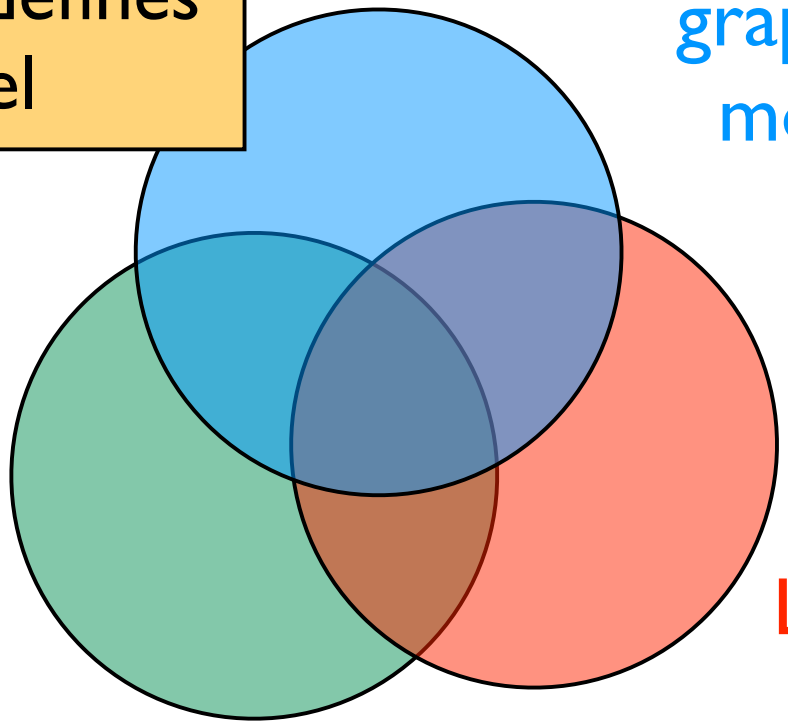
fixed set of random variables



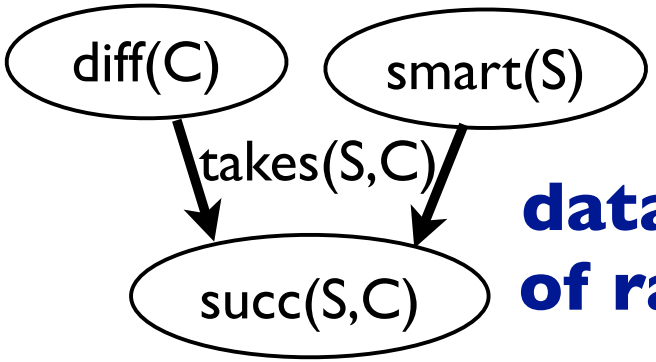
relational language defines graphical model

graphical model

relational definition of graphical model



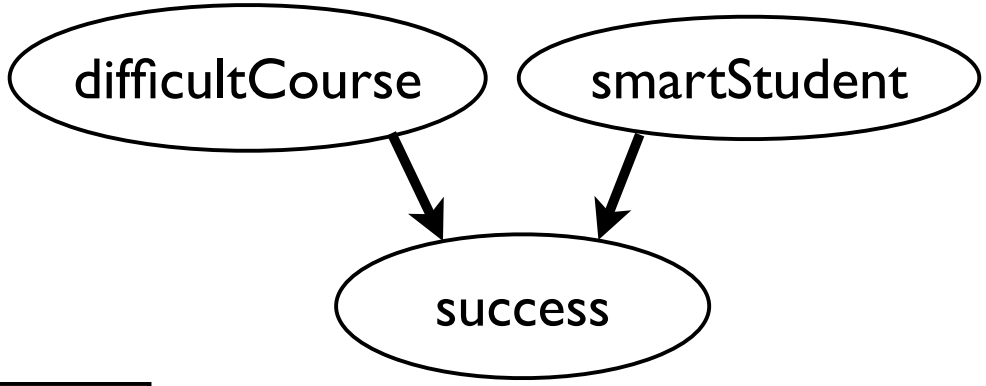
Learning



data-dependent set of random variables

Lifted graphical models

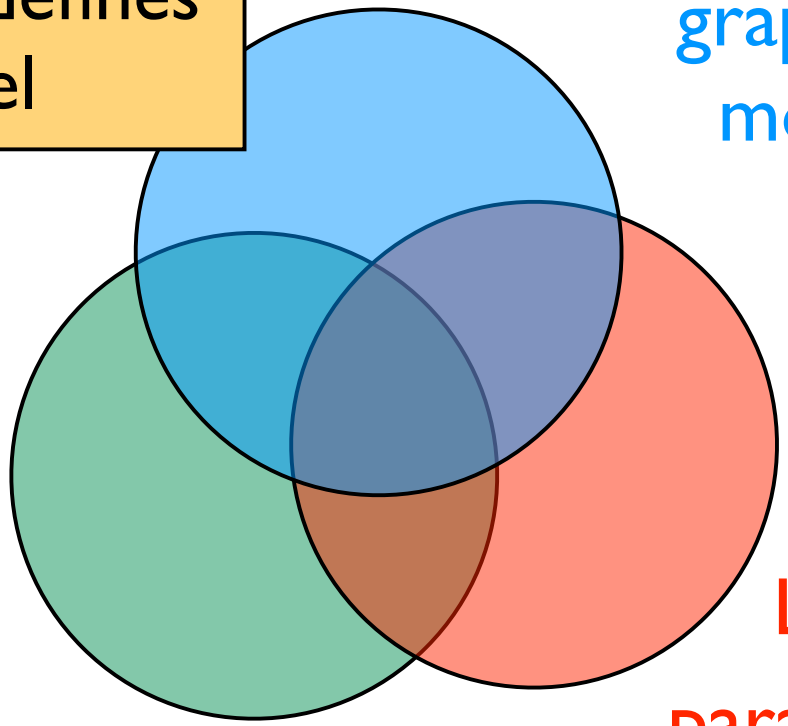
fixed set of random variables



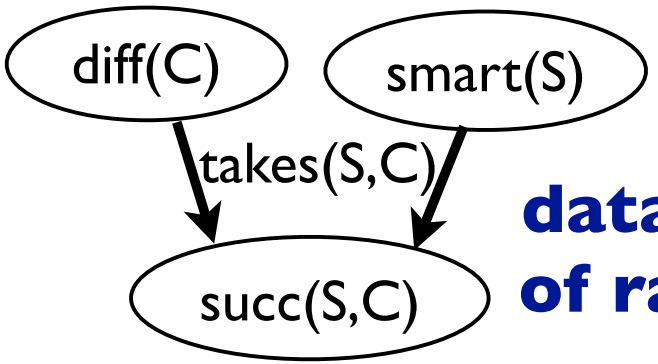
relational language defines graphical model

graphical model

relational definition of graphical model



Learning parameters & structure



data-dependent set of random variables

Lots of proposals in the literature, e.g.

- relational Markov networks (RMNs) [Taskar et al 2002]
- Markov logic networks (MLNs) [Richardson & Domingos 2006]
- probabilistic soft logic (PSL) [Broecheler et al 2010]
- FACTORIE [McCallum et al 2009]
- Bayesian logic programs (BLPs) [Kersting & De Raedt 2001]
- relational Bayesian networks (RBNs) [Jaeger 2002]
- logical Bayesian networks (LBNs) [Fierens et al 2005]
- probabilistic relational models (PRMs) [Koller & Pfeffer 1998]
- Bayesian logic (BLOG) [Milch et al 2005]
- CLP(BN) [Santos Costa et al 2008]
- probabilistic programming languages such as ProbLog, PRISM, Church, ...
- and many more ...

Lots of proposals in the literature, e.g.

- relational Markov networks (RMNs) [Taskar et al 2002]
- Markov logic networks (MLNs) [Richardson & Domingos 2006]
- probabilistic soft logic (PSL) [Broecheler et al 2010]
- FACTORIE [McCallum et al 2009]
- Bay
- rela
- logi
- probabilistic relational models (PRMs) [Koller & Pfeffer 1998]
- Bayesian logic (BLOG) [Milch et al 2005]
- CLP(BN) [Santos Costa et al 2008]
- probabilistic programming languages such as ProbLog, PRISM, Church, ...
- and many more ...

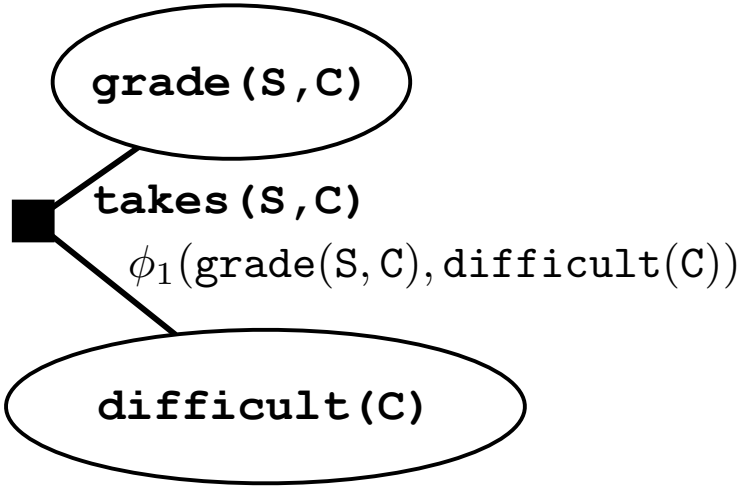
common principle:
parameterized factor graph
(par-factor graph)

Par-Factor Graph

[Poole 03]

Par-Factor Graph

[Poole 03]

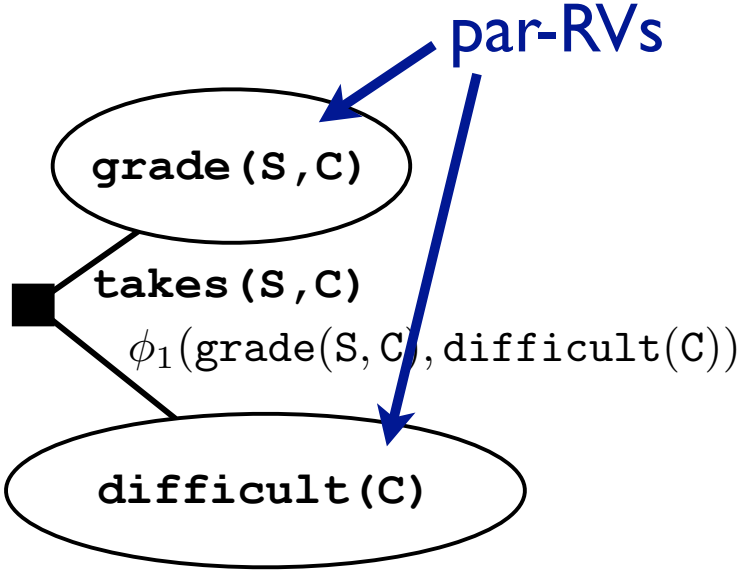


parameterized

factor (par-factor)

Par-Factor Graph

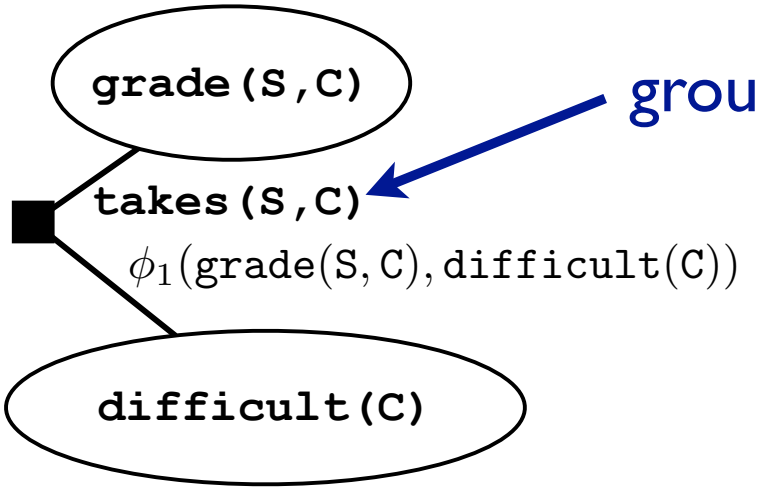
[Poole 03]



parameterized
factor (par-factor)

Par-Factor Graph

[Poole 03]



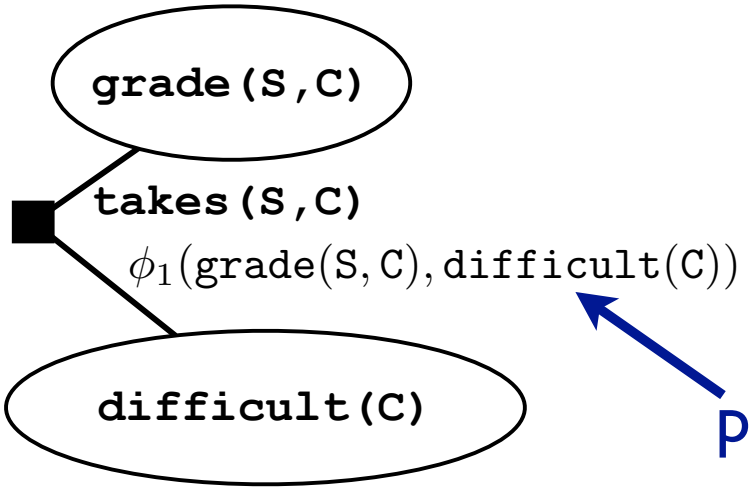
grounding constraint

parameterized

factor (par-factor)

Par-Factor Graph

[Poole 03]

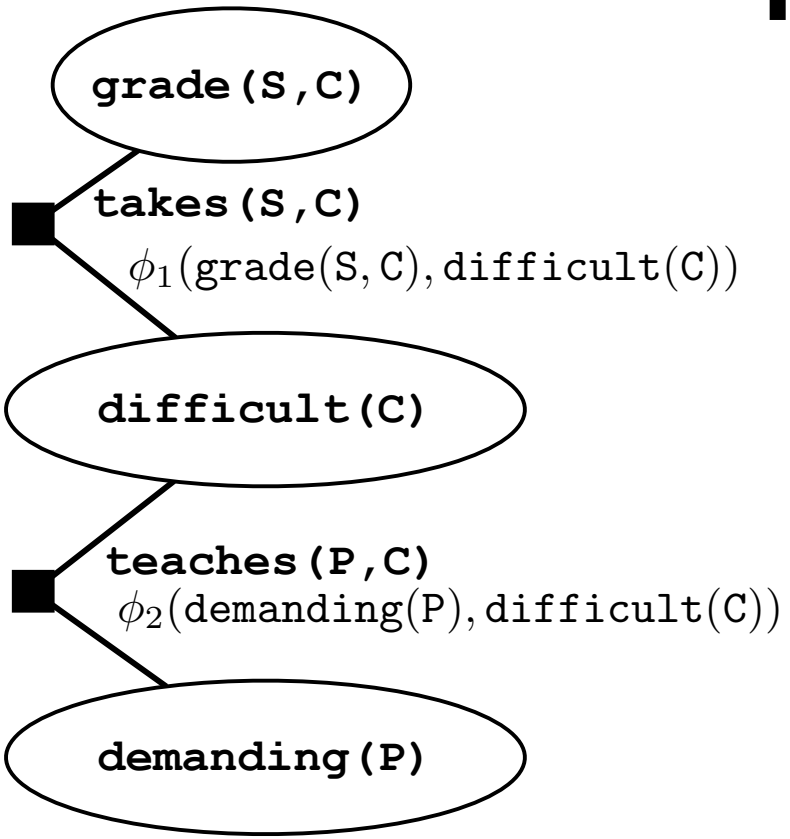


parameterized
factor (par-factor)

potential function

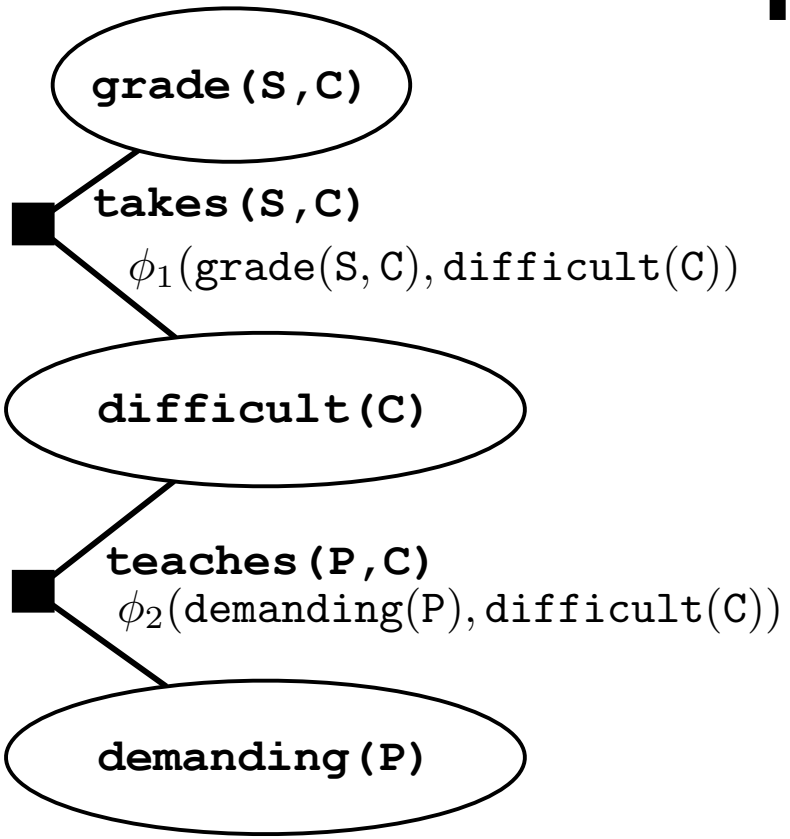
Par-Factor Graph

[Poole 03]



Par-Factor Graph

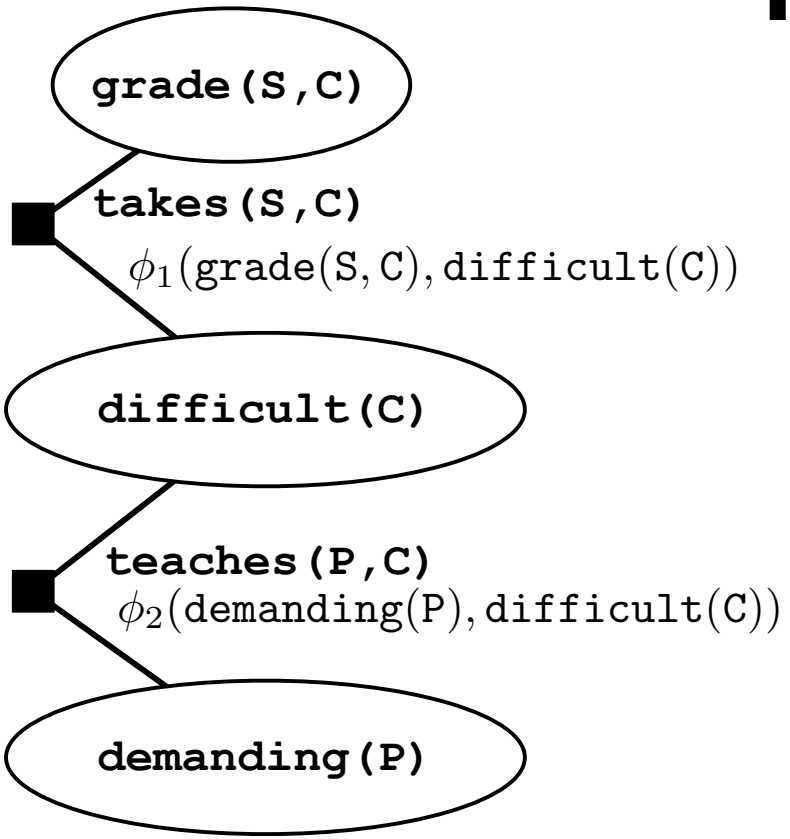
[Poole 03]



$$P(.) = \frac{1}{Z} \cdot \prod_{\texttt{takes(s,c)}} \phi_1(\texttt{grade}(\texttt{s},\texttt{c}),\texttt{difficult}(\texttt{c})) \cdot \prod_{\texttt{teaches(p,c)}} \phi_2(\texttt{demanding}(\texttt{p}),\texttt{difficult}(\texttt{c}))$$

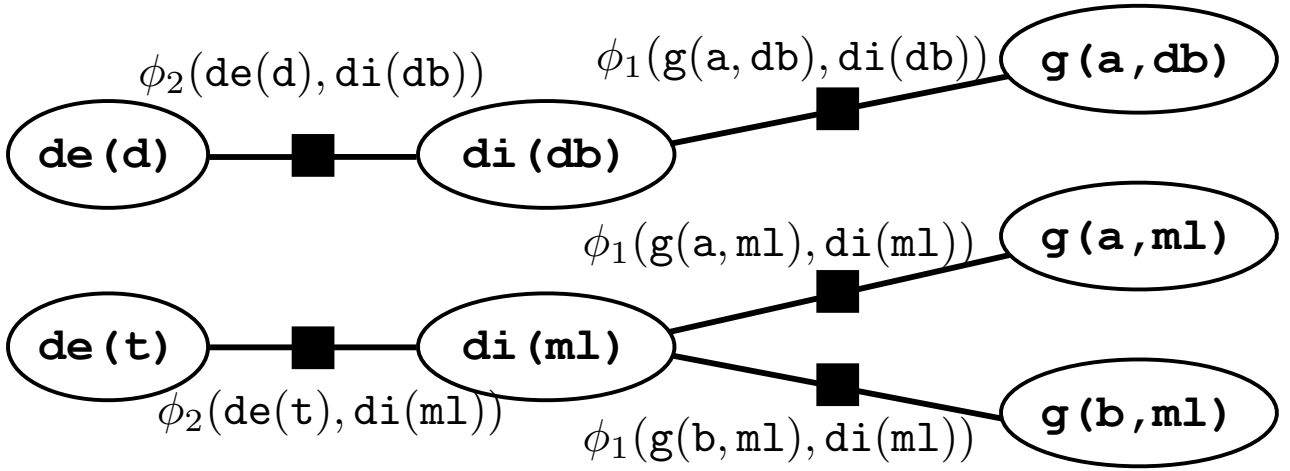
Par-Factor Graph

[Poole 03]

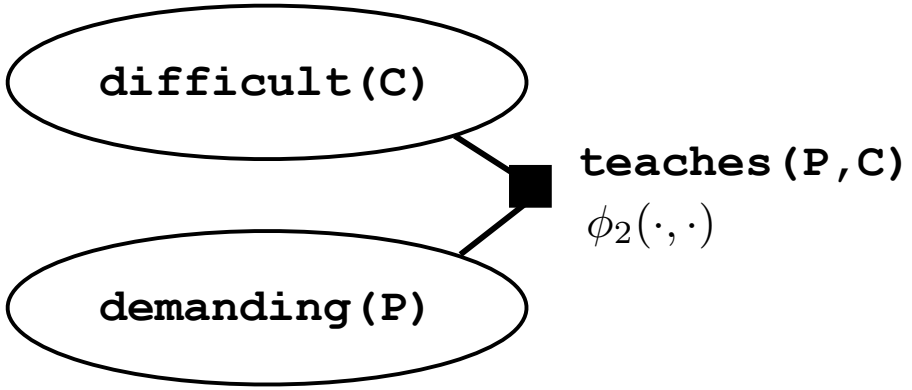


$$P(.) = \frac{1}{Z} \cdot \prod_{\texttt{takes(s,c)}} \phi_1(\texttt{grade}(\texttt{s},\texttt{c}),\texttt{difficult}(\texttt{c})) \cdot \prod_{\texttt{teaches(p,c)}} \phi_2(\texttt{demanding}(\texttt{p}),\texttt{difficult}(\texttt{c}))$$

```
takes(ann,ml).
takes(bob,ml).
takes(ann,db).
teaches(dan,db).
teaches(tom,ml).
```

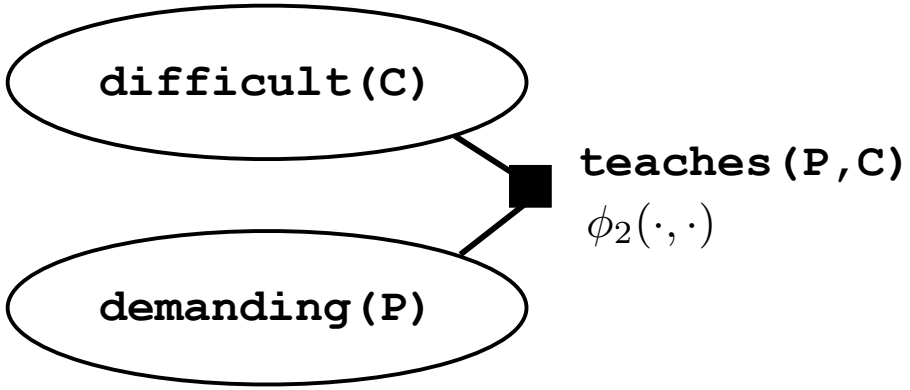


What if multiple professors teach the same course?




```
teaches(eve,db)
teaches(dan,db).
teaches(tom,ml).
teaches(bob,ai).
teaches(ann,ai).
teaches(ed,ai).
```

What if multiple professors teach the same course?

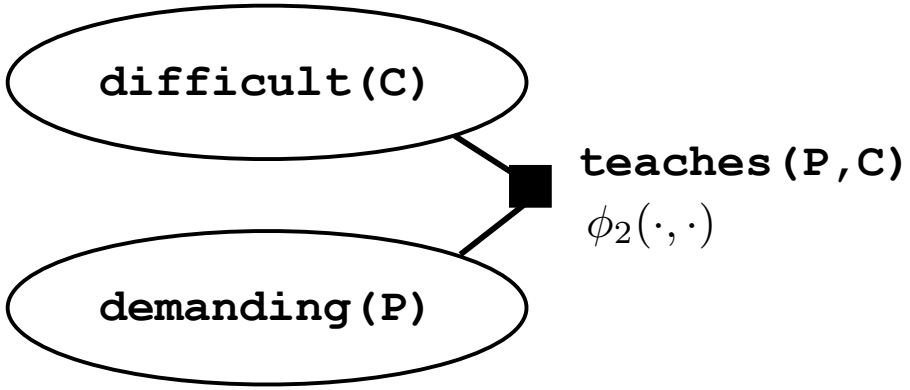



```
teaches(eve,db)
teaches(dan,db).
teaches(tom,ml).
teaches(bob,ai).
teaches(ann,ai).
teaches(ed,ai).
```

Option 1: aggregate values of RVs, then compute potential

$$\phi_2(\texttt{demanding(P)}, \texttt{difficult(C)}) = P(\texttt{difficult(C)}|\text{mean}\{\texttt{demanding(P)}\})$$

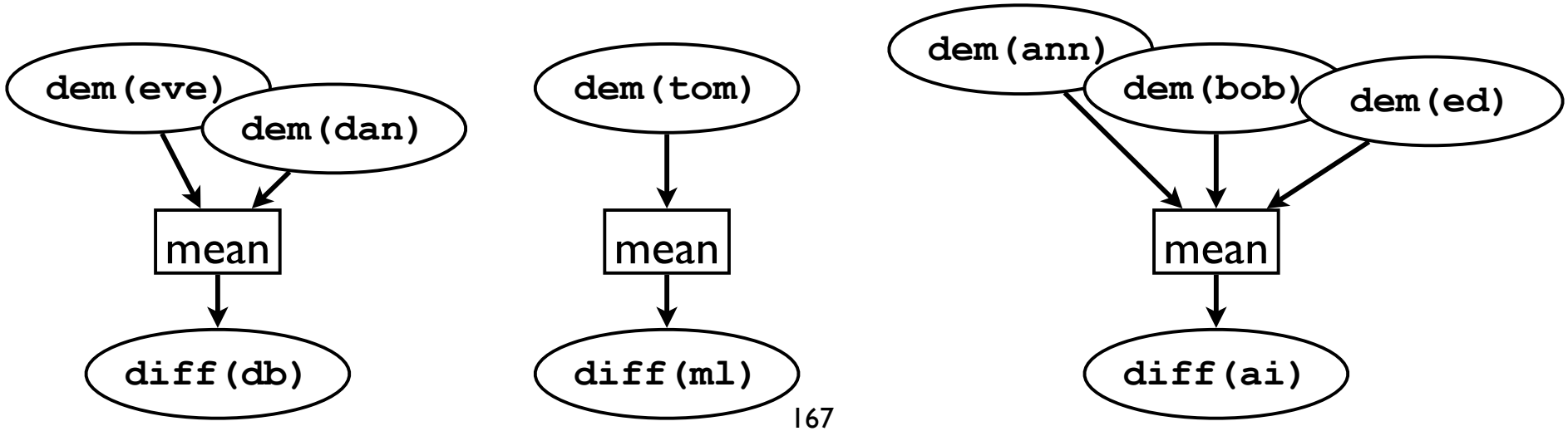
What if multiple professors teach the same course?



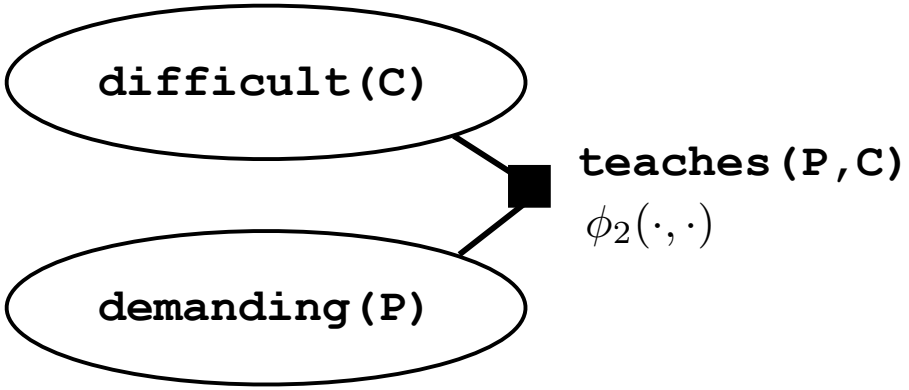

```
teaches(eve,db)
teaches(dan,db).
teaches(tom,ml).
teaches(bob,ai).
teaches(ann,ai).
teaches(ed,ai).
```

Option 1: aggregate values of RVs, then compute potential

$$\phi_2(\texttt{demanding(P)}, \texttt{difficult(C)}) = P(\texttt{difficult(C)}|\text{mean}\{\texttt{demanding(P)}\})$$



What if multiple professors teach the same course?

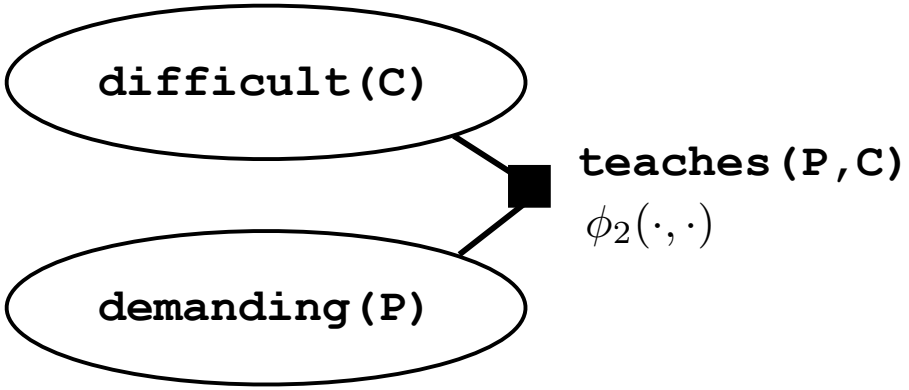



```
teaches(eve,db)
teaches(dan,db).
teaches(tom,ml).
teaches(bob,ai).
teaches(ann,ai).
teaches(ed,ai).
```

Option 2: compute potential for each RV, then combine

$$\phi_2(\texttt{demanding(P)}, \texttt{difficult(C)}) = 1 - \prod_{\texttt{P}} (1 - P(\texttt{difficult(C)} | \texttt{demanding(P)}))$$

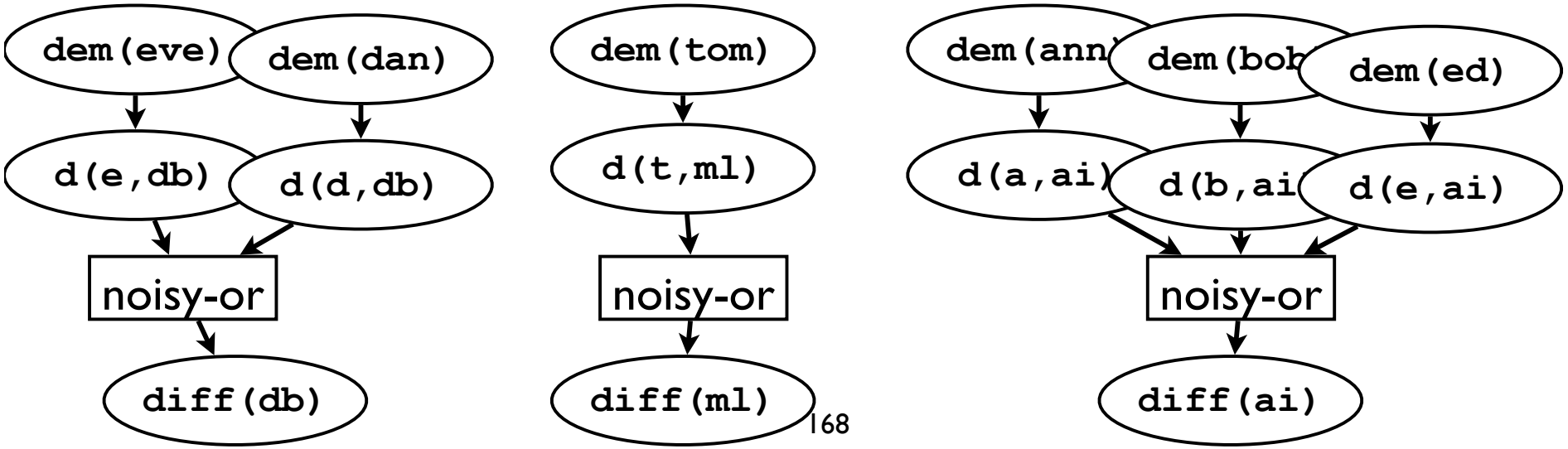
What if multiple professors teach the same course?




```
teaches(eve,db)
teaches(dan,db).
teaches(tom,ml).
teaches(bob,ai).
teaches(ann,ai).
teaches(ed,ai).
```

Option 2: compute potential for each RV, then combine

$$\phi_2(\texttt{demanding(P)}, \texttt{difficult(C)}) = 1 - \prod_{\texttt{P}} (1 - P(\texttt{difficult(C)} | \texttt{demanding(P)}))$$



Example: CLP(BN)

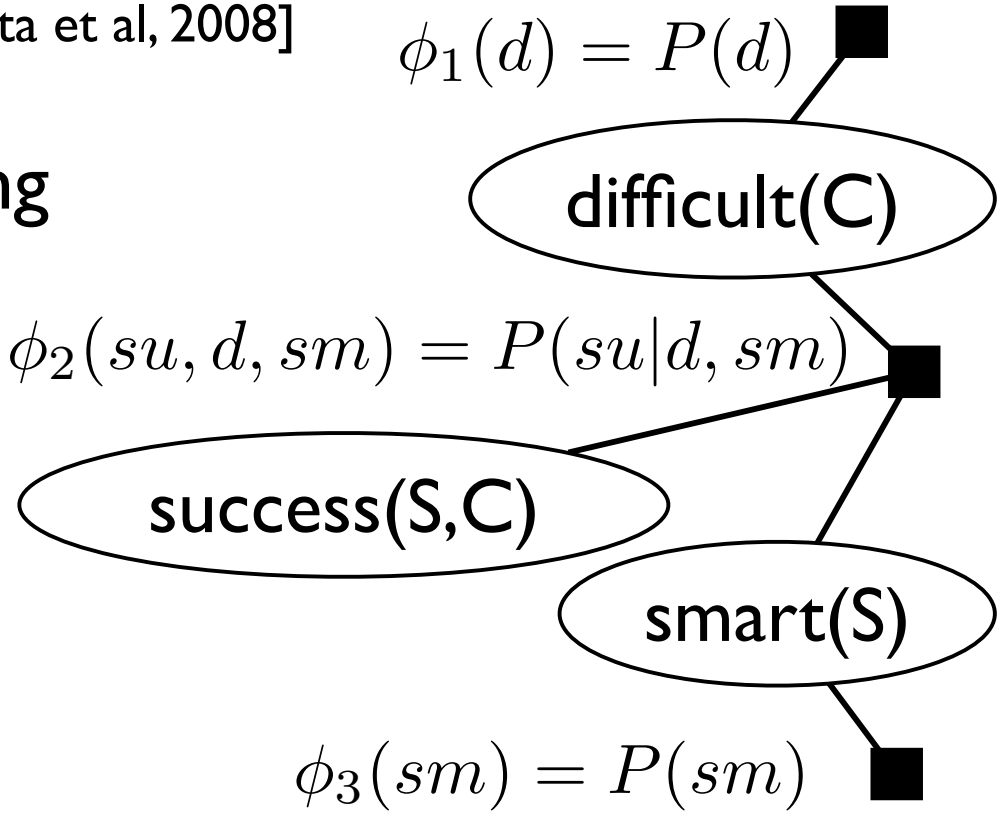
CLP(BN) [Santos Costa et al, 2008]

constraint logic programming
for Bayesian networks

CLP(BN)

[Santos Costa et al, 2008]

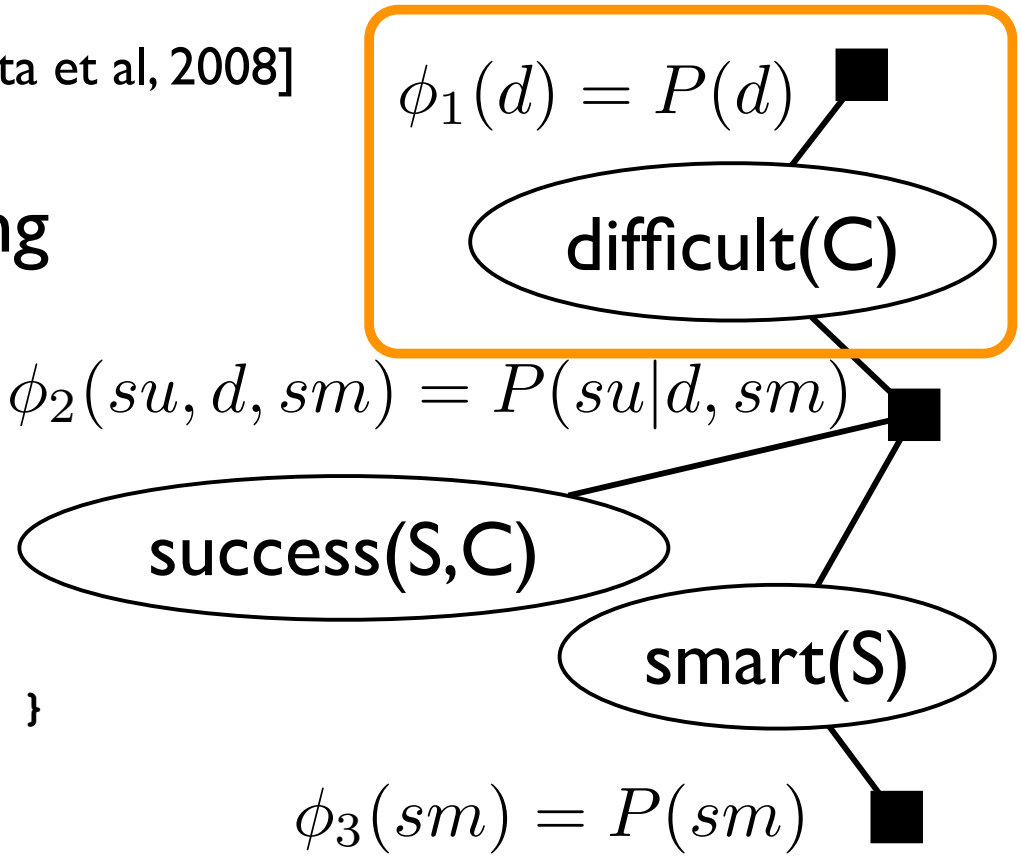
constraint logic programming
for Bayesian networks



CLP(BN)

[Santos Costa et al, 2008]

constraint logic programming for Bayesian networks

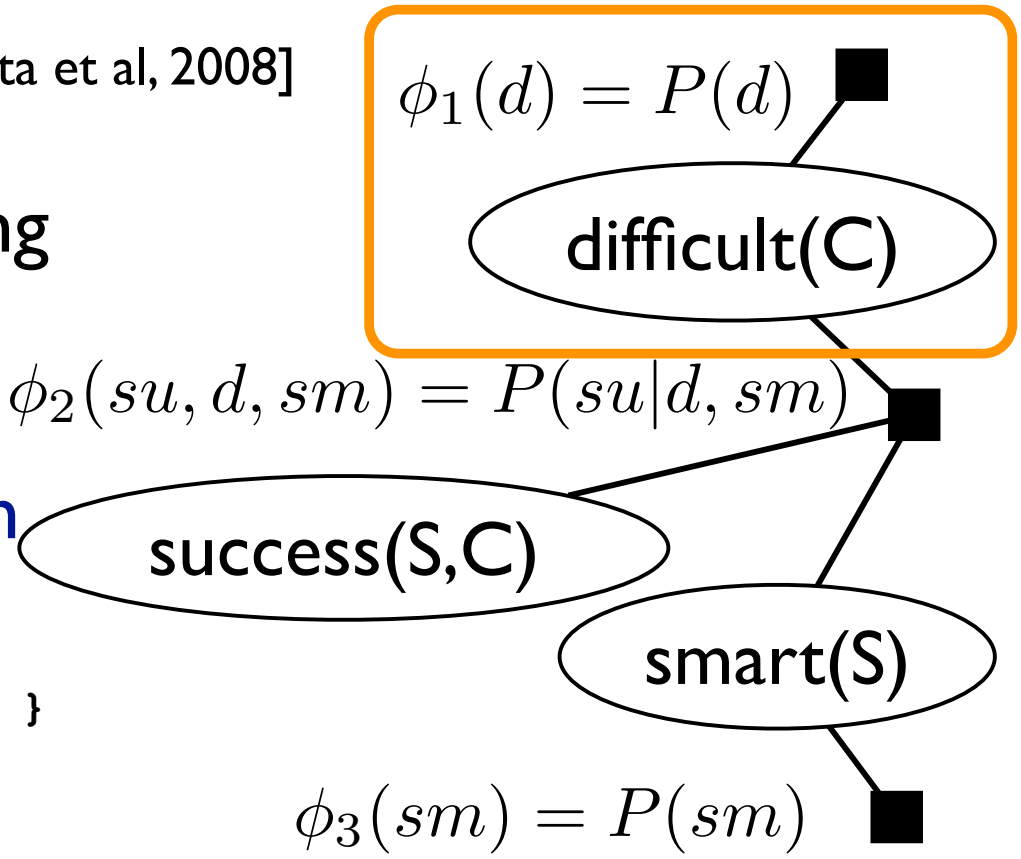



```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }
```

CLP(BN)

[Santos Costa et al, 2008]

constraint logic programming for Bayesian networks



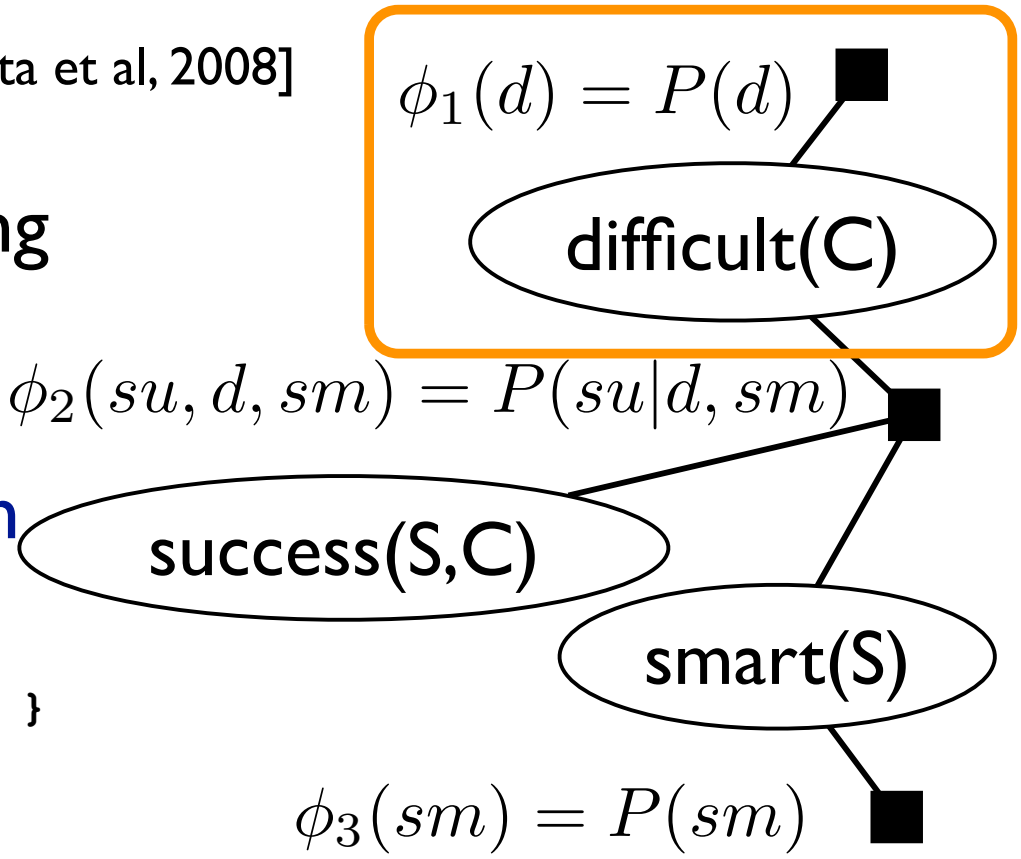
par-RV: unique skolem term

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }
```

CLP(BN)

[Santos Costa et al, 2008]

constraint logic programming
for Bayesian networks



par-RV: unique skolem term

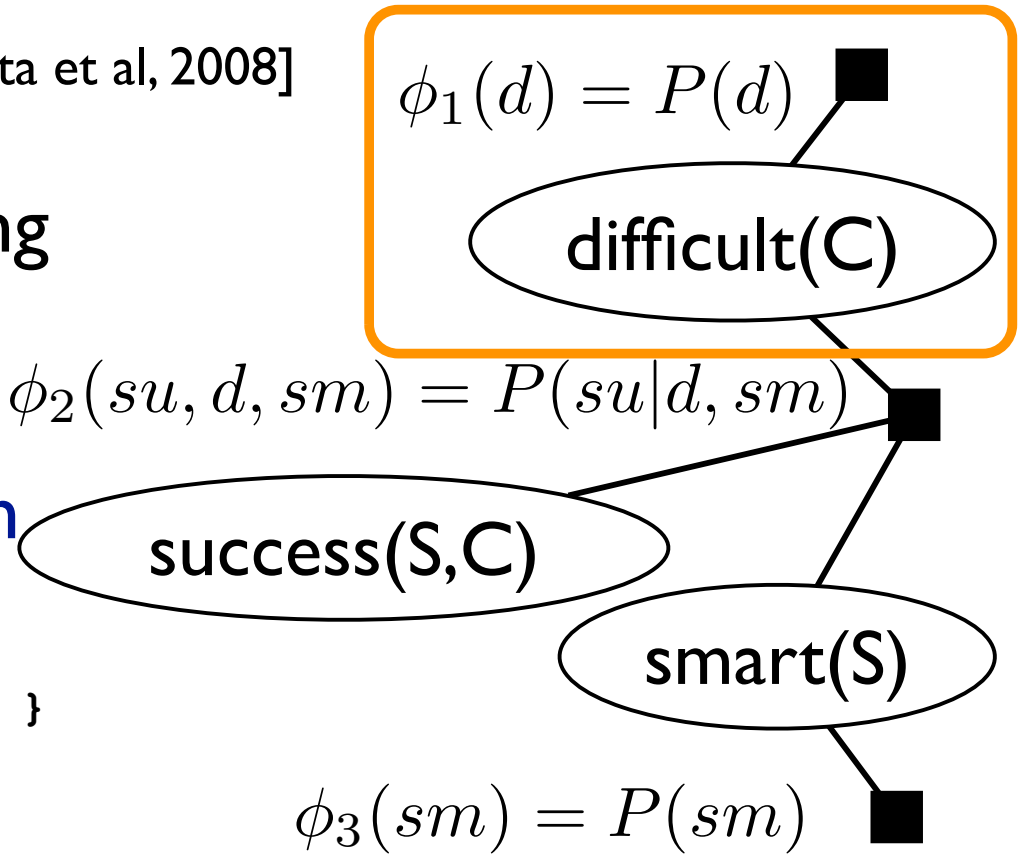
```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }
```

possible values

CLP(BN)

[Santos Costa et al, 2008]

constraint logic programming
for Bayesian networks



par-RV: unique skolem term

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }
```

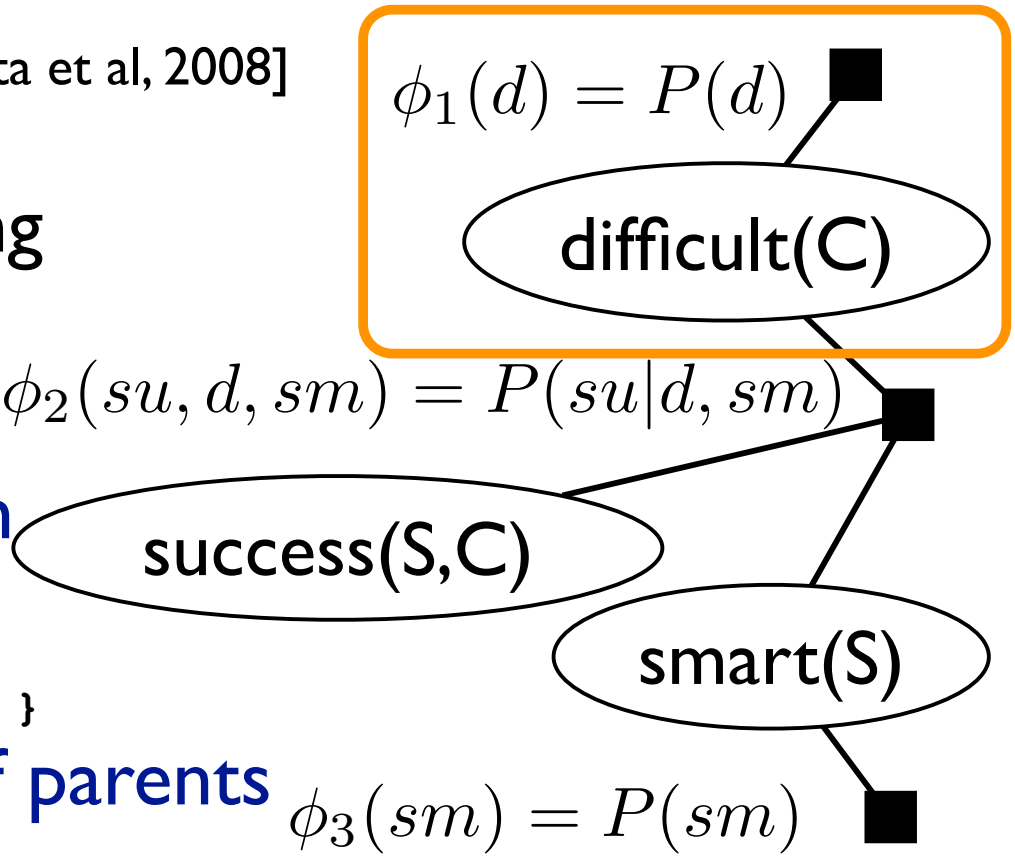
possible values

their probabilities

CLP(BN)

[Santos Costa et al, 2008]

constraint logic programming
for Bayesian networks



par-RV: unique skolem term

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }
```

possible values

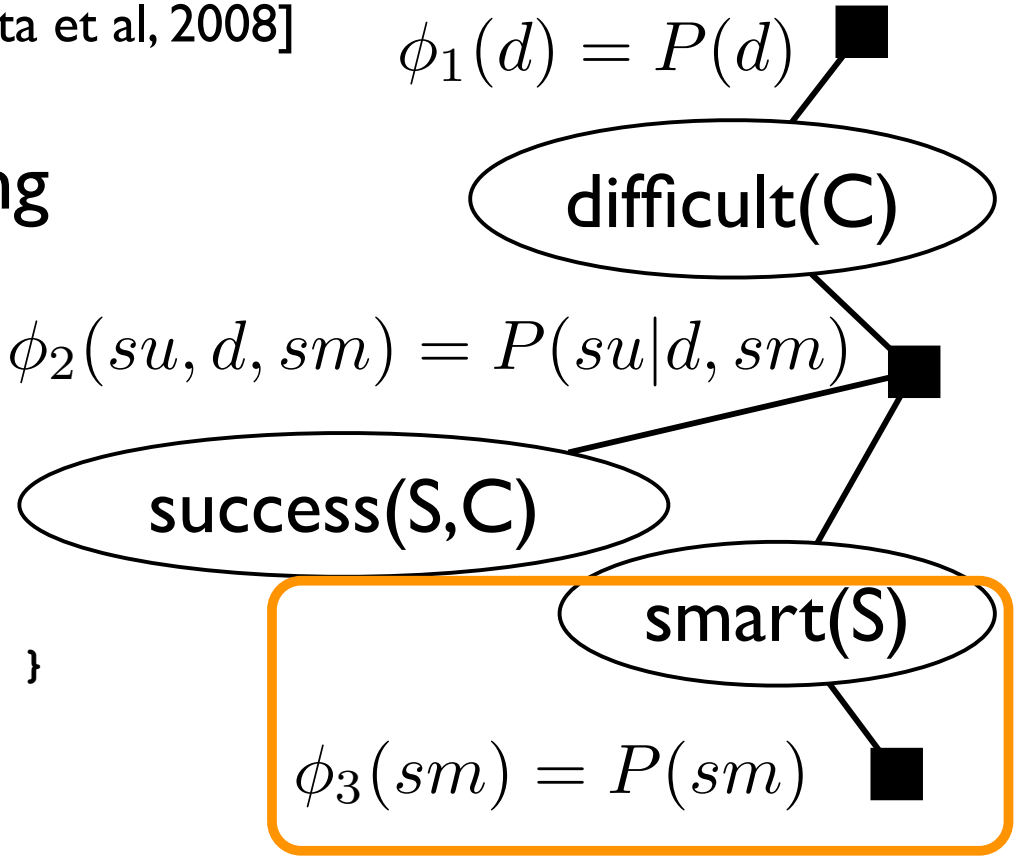
their probabilities

list of parents

CLP(BN)

[Santos Costa et al, 2008]

constraint logic programming for Bayesian networks

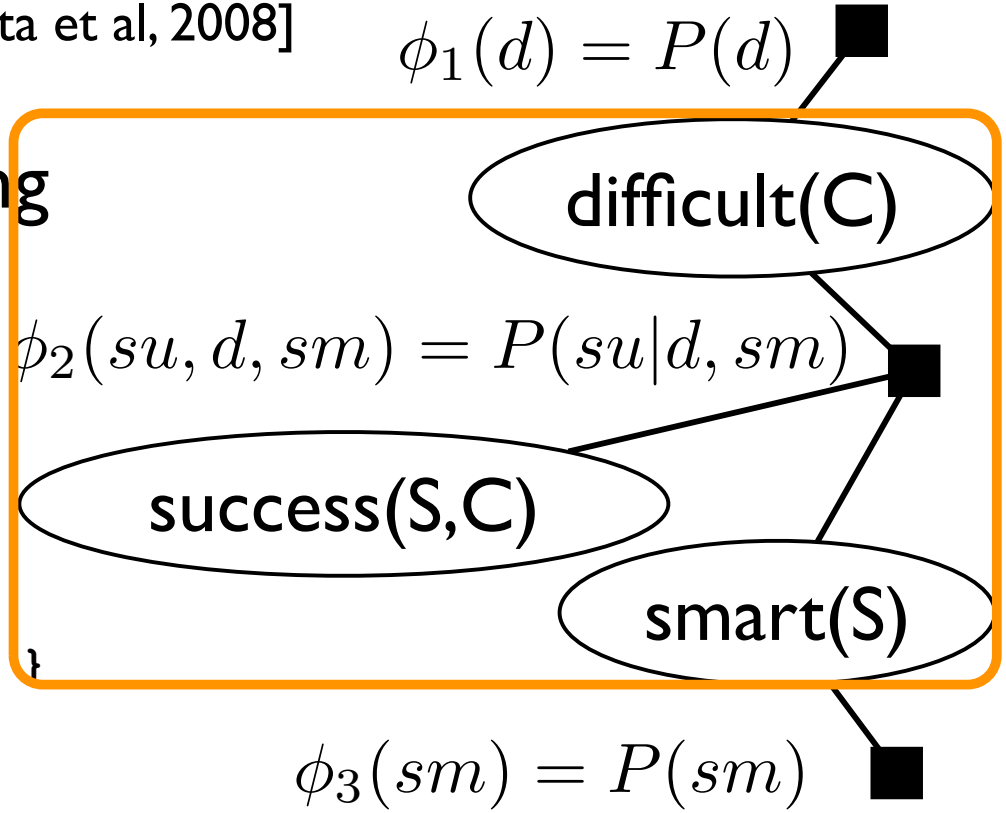



```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[]) }
```

CLP(BN)

[Santos Costa et al, 2008]

constraint logic programming for Bayesian networks

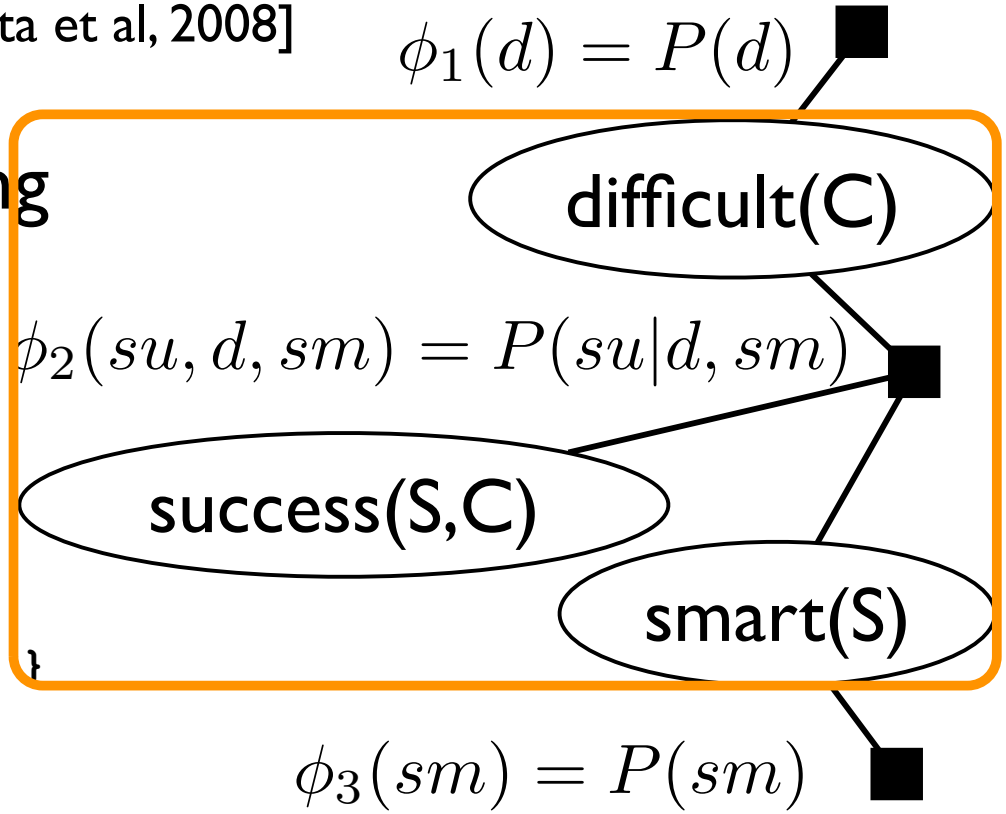



```
difficult(Course,Diff) :-
     { Diff = d(Course)
       with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
     { Smart = sm(Student)
       with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
     takes(Student,Course),
     difficult(Course,D),
     smart(Student,Sm),
     { Success = su(Student,Course)
       with p([t,f],[0.85,0.1,0.98,0.45,
                     0.15,0.9,0.02,0.55],[D,Sm])}
```

CLP(BN)

[Santos Costa et al, 2008]

constraint logic programming for Bayesian networks



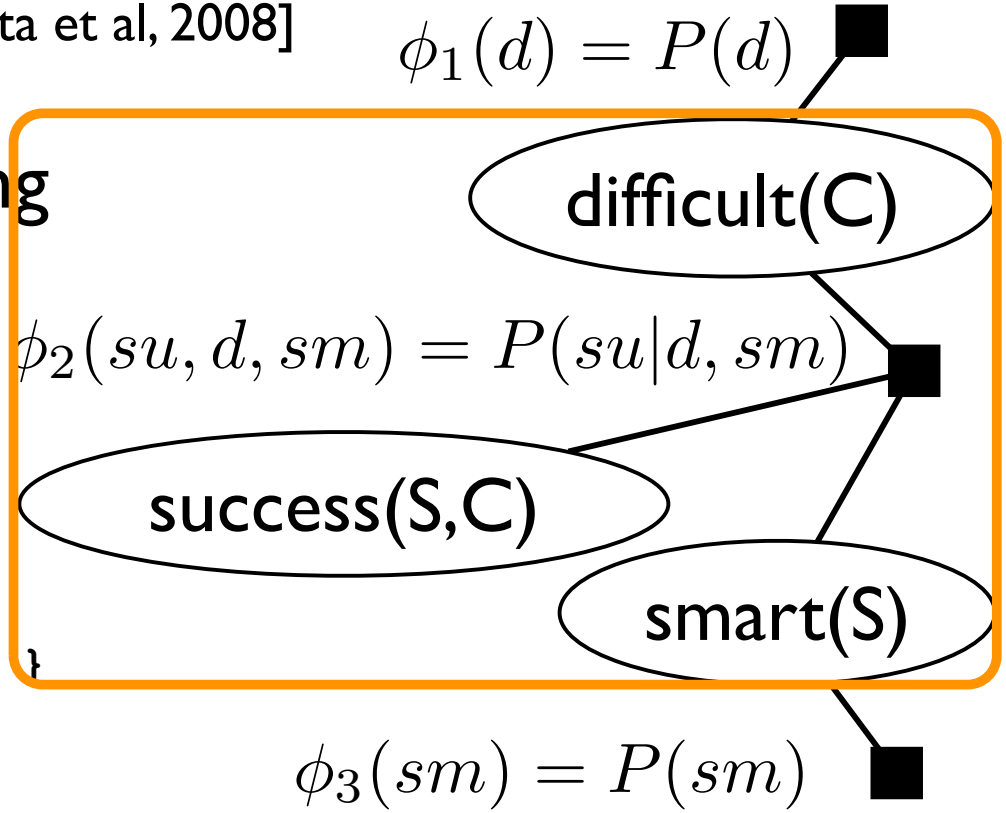

```
difficult(Course,Diff) :-
     { Diff = d(Course)
       with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
     { Smart = sm(Student)
       with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
     takes(Student,Course),
     difficult(Course,D),
     smart(Student,Sm),
     { Success = su(Student,Course)
       with p([t,f],[0.85,0.1,0.98,0.45,
                     0.15,0.9,0.02,0.55],[D,Sm])}
```

Prolog call constrains grounding

CLP(BN)

[Santos Costa et al, 2008]

constraint logic programming for Bayesian networks



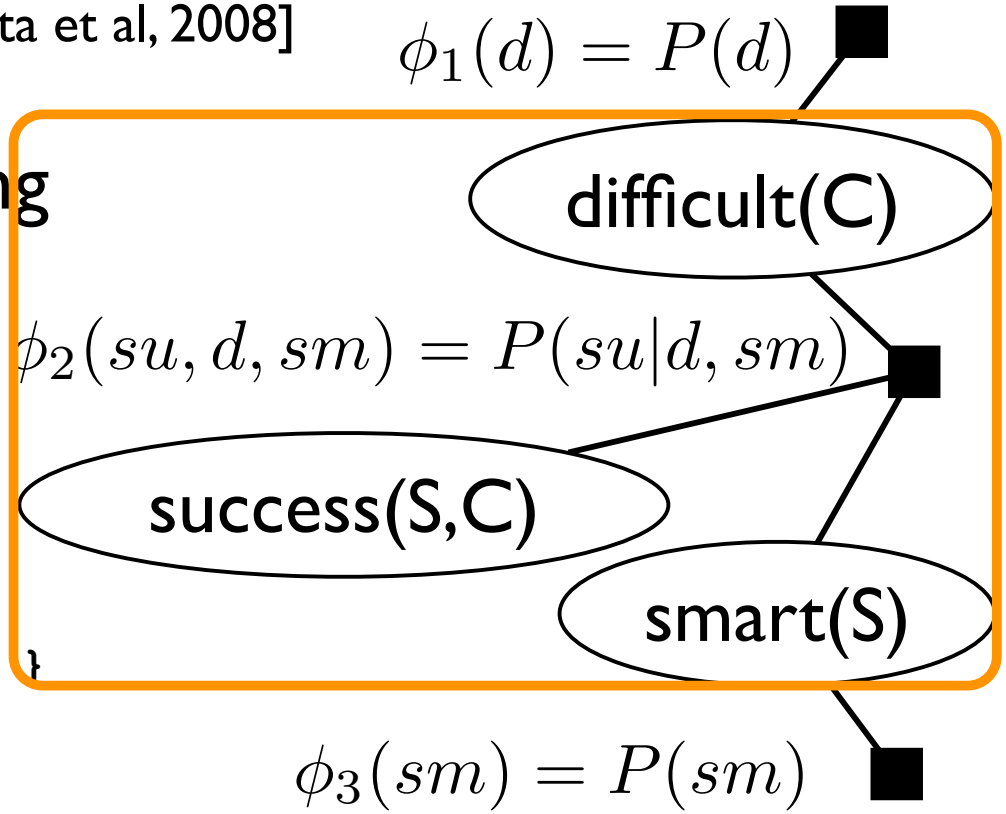

```
difficult(Course,Diff) :-
     { Diff = d(Course)
       with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
     { Smart = sm(Student)
       with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
     takes(Student,Course),
     difficult(Course,D),
     smart(Student,Sm),
     { Success = su(Student,Course)
       with p([t,f],[0.85,0.1,0.98,0.45,
                     0.15,0.9,0.02,0.55],[D,Sm]) }
```

get parents

CLP(BN)

[Santos Costa et al, 2008]

constraint logic programming for Bayesian networks



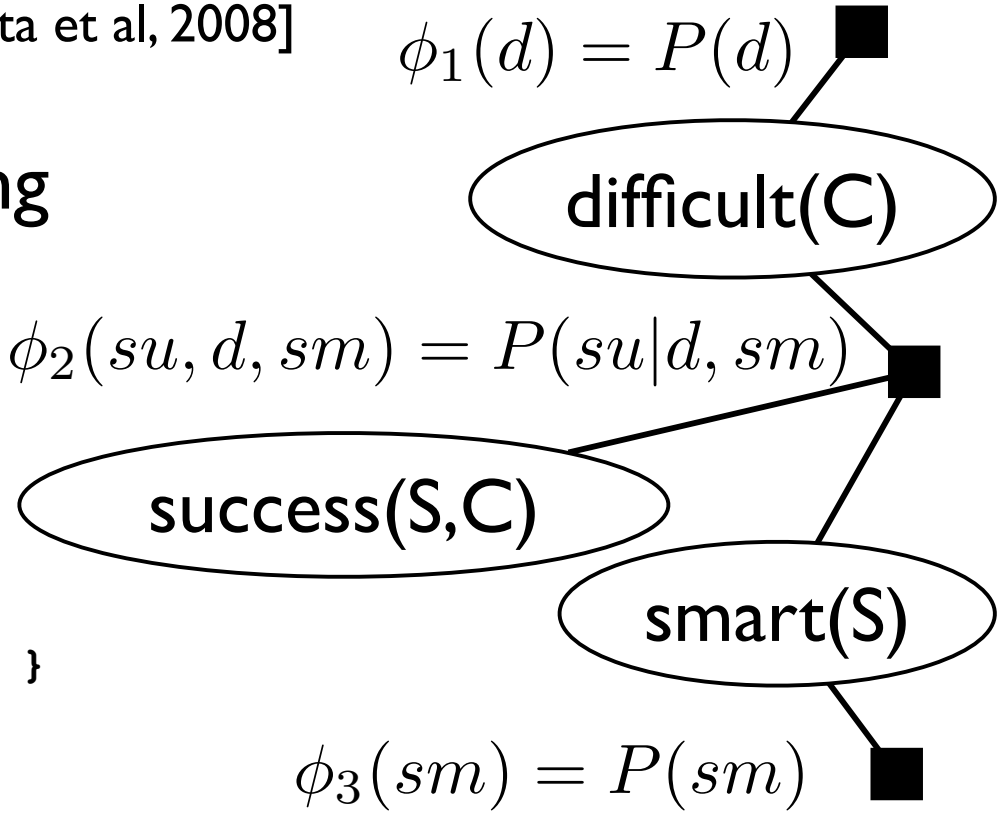

```
difficult(Course,Diff) :-
     { Diff = d(Course)
       with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
     { Smart = sm(Student)
       with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
     takes(Student,Course),
     difficult(Course,D),
     smart(Student,Sm),
     { Success = su(Student,Course)
       with p([t,f],[0.85,0.1,0.98,0.45,
                     0.15,0.9,0.02,0.55],[D,Sm]) }
```

get parents

P(su|d=t,sm=t)

CLP(BN) [Santos Costa et al, 2008]

constraint logic programming for Bayesian networks

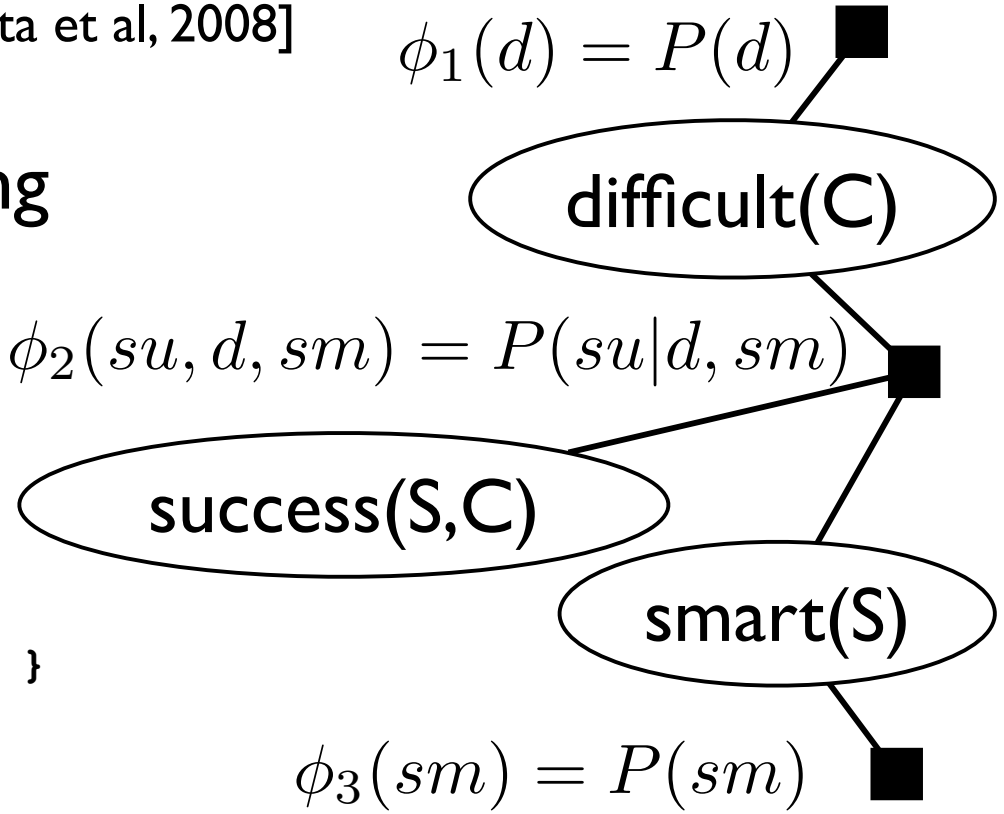



```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
    takes(Student,Course),
    difficult(Course,D),
    smart(Student,Sm),
    { Success = su(Student,Course)
      with p([t,f],[0.85,0.1,0.98,0.45,
                    0.15,0.9,0.02,0.55],[D,Sm])}
```

CLP(BN)

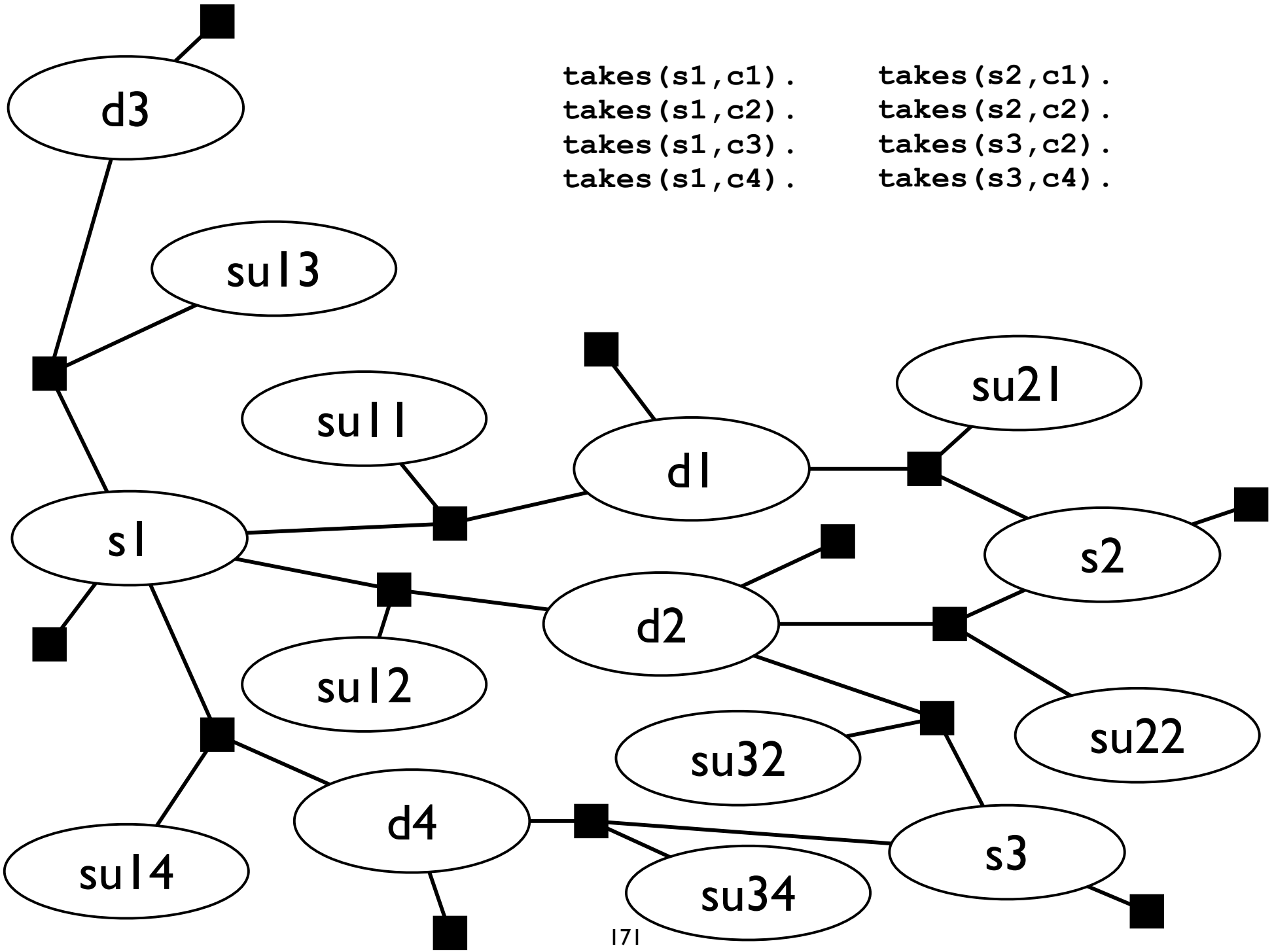
[Santos Costa et al, 2008]

constraint logic programming for Bayesian networks




```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
    takes(Student,Course),
    difficult(Course,D),
    smart(Student,Sm),
    { Success = su(Student,Course)
      with p([t,f],[0.85,0.1,0.98,0.45,
                    0.15,0.9,0.02,0.55],[D,Sm])}
```

```
takes(s1,c1).
takes(s1,c2).
takes(s1,c3).
takes(s1,c4).
takes(s2,c1).
takes(s2,c2).
takes(s3,c2).
takes(s3,c4).
```


takes(s1,c1).
takes(s1,c2).
takes(s1,c3).
takes(s1,c4).
takes(s2,c1).
takes(s2,c2).
takes(s3,c2).
takes(s3,c4).
d3
su13
su11
d1
su21
s1
s2
d2
su12
su22
su32
d4
s3
su14
su34

in ProbLog?

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
    takes(Student,Course),
    difficult(Course,D),
    smart(Student,Sm),
    { Success = su(Student,Course)
      with p([t,f],[0.85,0.1,0.98,0.45
                    0.15,0.9,0.02,0.55],[D,Sm])}
```

in ProbLog?

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
    takes(Student,Course),
    difficult(Course,D),
    smart(Student,Sm),
    { Success = su(Student,Course)
      with p([t,f],[0.85,0.1,0.98,0.45
                    0.15,0.9,0.02,0.55],[D,Sm])}
```

```
0.6::difficult(C) :- takes(_,C).
0.7::smart(S) :- takes(S,_).
0.85::success(S,C) <- takes(S,C),difficult(C), smart(S).
0.10::success(S,C) <- takes(S,C),difficult(C), \+smart(S).
0.98::success(S,C) <- takes(S,C),\+difficult(C), smart(S).
0.45::success(S,C) <- takes(S,C),\+difficult(C), \+smart(S).
```

Inference Example

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
    takes(Student,Course),
    difficult(Course,D),
    smart(Student,Sm),
    { Success = su(Student,Course)
      with p([t,f],[0.85,0.1,0.98,0.45,
                    0.15,0.9,0.02,0.55],[D,Sm])}
```

```
takes(s1,c1).
takes(s1,c2).
takes(s2,c1).
takes(s2,c2).
takes(s3,c2).
takes(s3,c4).
```

Inference Example

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
    takes(Student,Course),
    difficult(Course,D),
    smart(Student,Sm),
    { Success = su(Student,Course)
      with p([t,f],[0.85,0.1,0.98,0.45,
                    0.15,0.9,0.02,0.55],[D,Sm])}
```

```
takes(s1,c1).
takes(s1,c2).
takes(s2,c1).
takes(s2,c2).
takes(s3,c2).
takes(s3,c4).
```

```
?-success(s2,c1,S).
```

Inference Example

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
    takes(Student,Course),
    difficult(Course,D),
    smart(Student,Sm),
    { Success = su(Student,Course)
      with p([t,f],[0.85,0.1,0.98,0.45,
                    0.15,0.9,0.02,0.55],[D,Sm])}
```

```
takes(s1,c1).
takes(s1,c2).
takes(s2,c1).
takes(s2,c2).
takes(s3,c2).
takes(s3,c4).
```

```
?-success(s2,c1,S).
```

Inference Example

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
    takes(Student,Course),
    difficult(Course,D)
    smart(Student,Sm),
    { Success = su(Student,Course)
      with p([t,f],[0.85,0.1,0.98,0.45,
                    0.15,0.9,0.02,0.55],[D,Sm])}
```

```
takes(s1,c1).
takes(s1,c2).
takes(s2,c1).
takes(s2,c2).
takes(s3,c2).
takes(s3,c4).
```

```
?-success(s2,c1,S).
```

Inference Example

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
    takes(Student,Course),
    difficult(Course,D)
    smart(Student,Sm),
    { Success = su(Student,Course)
      with p([t,f],[0.85,0.1,0.98,0.45,
                    0.15,0.9,0.02,0.55],[D,Sm])}
```

```
takes(s1,c1).
takes(s1,c2).
takes(s2,c1).
takes(s2,c2).
takes(s3,c2).
takes(s3,c4).
```

```
?-success(s2,c1,S).
```

D=d(c1) with p(...,[])

Inference Example

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
    takes(Student,Course),
    difficult(Course,D),
    smart(Student,Sm),
    { Success = su(Student,Course)
      with p([t,f],[0.85,0.1,0.98,0.45,
                    0.15,0.9,0.02,0.55],[D,Sm])}
```

```
takes(s1,c1).
takes(s1,c2).
takes(s2,c1).
takes(s2,c2).
takes(s3,c2).
takes(s3,c4).
```

```
?-success(s2,c1,S).
```

D=d(c1) with p(...,[])

Inference Example

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
    takes(Student,Course),
    difficult(Course,D),
    smart(Student,Sm),
    { Success = su(Student,Course)
      with p([t,f],[0.85,0.1,0.98,0.45,
                    0.15,0.9,0.02,0.55],[D,Sm])}
```

```
takes(s1,c1).
takes(s1,c2).
takes(s2,c1).
takes(s2,c2).
takes(s3,c2).
takes(s3,c4).
```

```
?-success(s2,c1,S).
```

Sm=sm(s2) with p(...,[])

D=d(c1) with p(...,[])

Inference Example

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
    takes(Student,Course),
    difficult(Course,D),
    smart(Student,Sm),
    { Success = su(Student,Course)
      with p([t,f],[0.85,0.1,0.98,0.45,
                    0.15,0.9,0.02,0.55],[D,Sm])}
```

```
takes(s1,c1).
takes(s1,c2).
takes(s2,c1).
takes(s2,c2).
takes(s3,c2).
takes(s3,c4).
```

```
?-success(s2,c1,S).
```

Sm=sm(s2) with p(...,[])

D=d(c1) with p(...,[])

Inference Example

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
    takes(Student,Course),
    difficult(Course,D),
    smart(Student,Sm),
    { Success = su(Student,Course)
      with p([t,f],[0.85,0.1,0.98,0.45,
                    0.15,0.9,0.02,0.55],[D,Sm])}
```

```
takes(s1,c1).
takes(s1,c2).
takes(s2,c1).
takes(s2,c2).
takes(s3,c2).
takes(s3,c4).
```

```
?-success(s2,c1,S).
```

Success=su(s2,c1) with p(...,[D,Sm])

Sm=sm(s2) with p(...,[])

D=d(c1) with p(...,[])

Inference Example

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
    takes(Student,Course),
    difficult(Course,D),
    smart(Student,Sm),
    { Success = su(Student,Course)
      with p([t,f],[0.85,0.1,0.98,0.45,
                    0.15,0.9,0.02,0.55],[D,Sm])}
```

```
takes(s1,c1).
takes(s1,c2).
takes(s2,c1).
takes(s2,c2).
takes(s3,c2).
takes(s3,c4).
```

```
?-success(s2,c1,S).
```

Success=su(s2,c1) with p(...,[D,Sm])

Sm=sm(s2) with p(...,[])

D=d(c1) with p(...,[])

Inference Example

```
difficult(Course,Diff) :-
    { Diff = d(Course)
      with p([t,f],[0.6,0.4],[]) }

smart(Student,Smart) :-
    { Smart = sm(Student)
      with p([t,f],[0.7,0.3],[]) }

success(Student,Course,Success) :-
    takes(Student,Course),
    difficult(Course,D),
    smart(Student,Sm),
    { Success = su(Student,Course)
      with p([t,f],[0.85,0.1,0.98,0.45,
                    0.15,0.9,0.02,0.55],[D,Sm])}
```

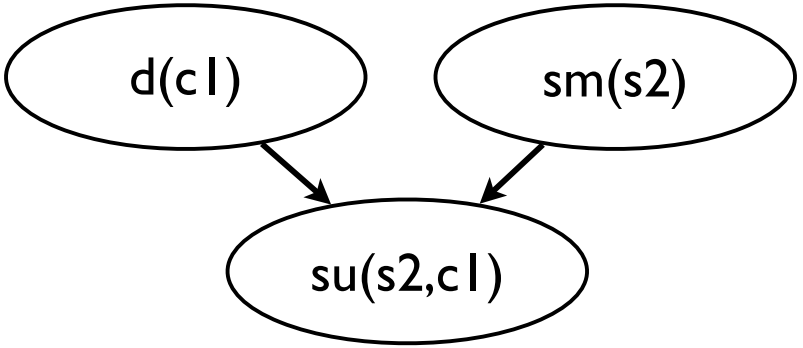
```
takes(s1,c1).
takes(s1,c2).
takes(s2,c1).
takes(s2,c2).
takes(s3,c2).
takes(s3,c4).
```

```
?-success(s2,c1,S).
```

Success=su(s2,c1) with p(...,[D,Sm])

Sm=sm(s2) with p(...,[])

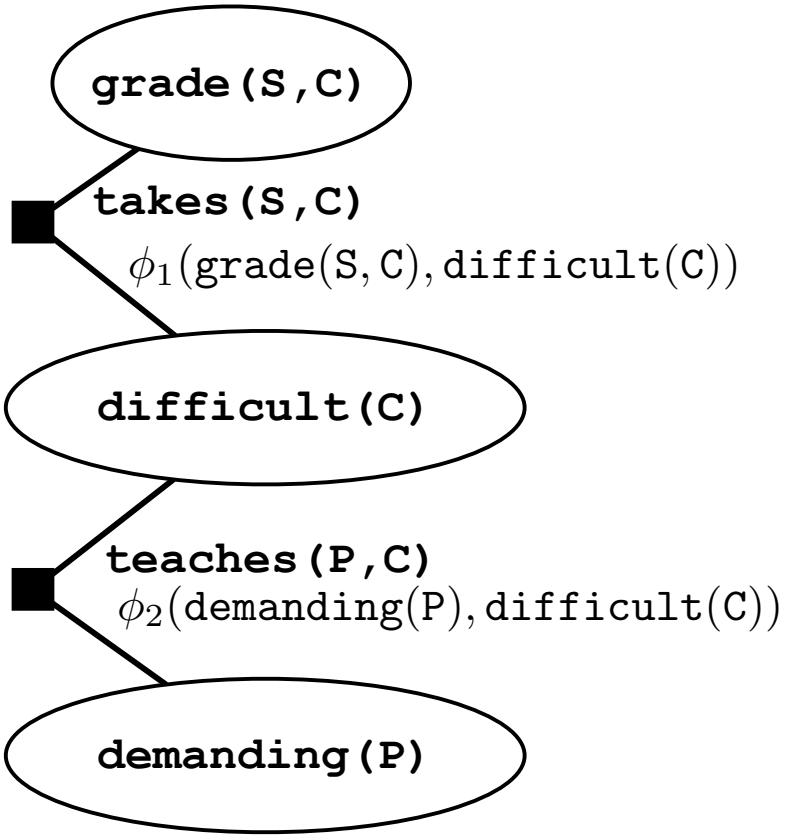
D=d(c1) with p(...,[])



CLP(BN) Summary

- Templating Bayesian networks via constraint logic programming
- Knowledge-based model construction (KBMC):
  - construct relevant ground BN by backward reasoning, adding constraints to constraint store
  - run any propositional inference technique

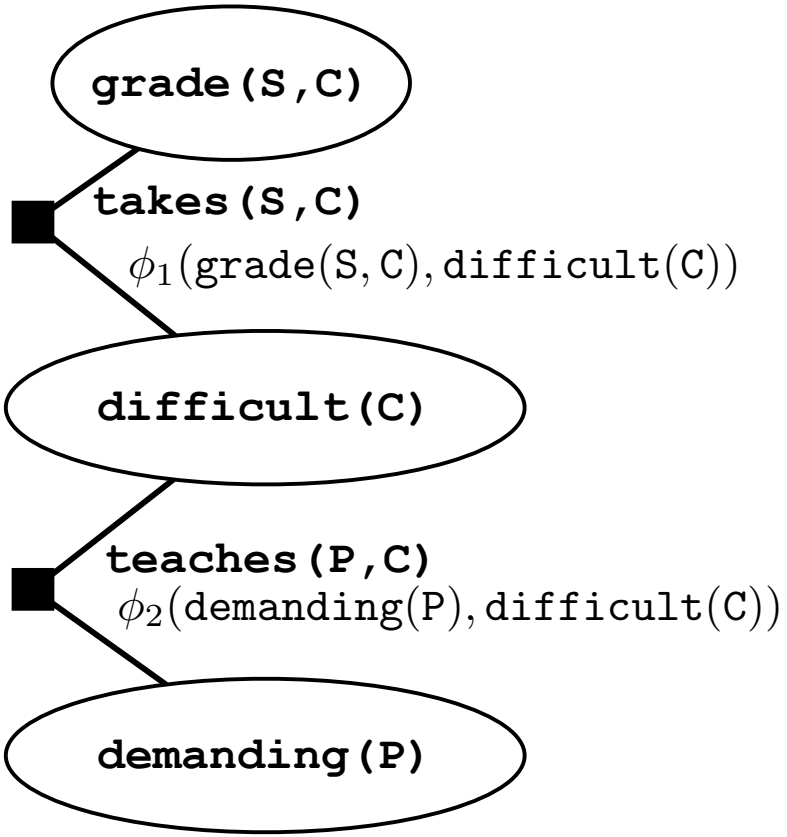
Inference by Grounding



demanding(tom)?

```
 takes(ann,ml).
 takes(bob,ml).
 takes(ann,db).
teaches(dan,db).
teaches(tom,ml).
```

Inference by Grounding

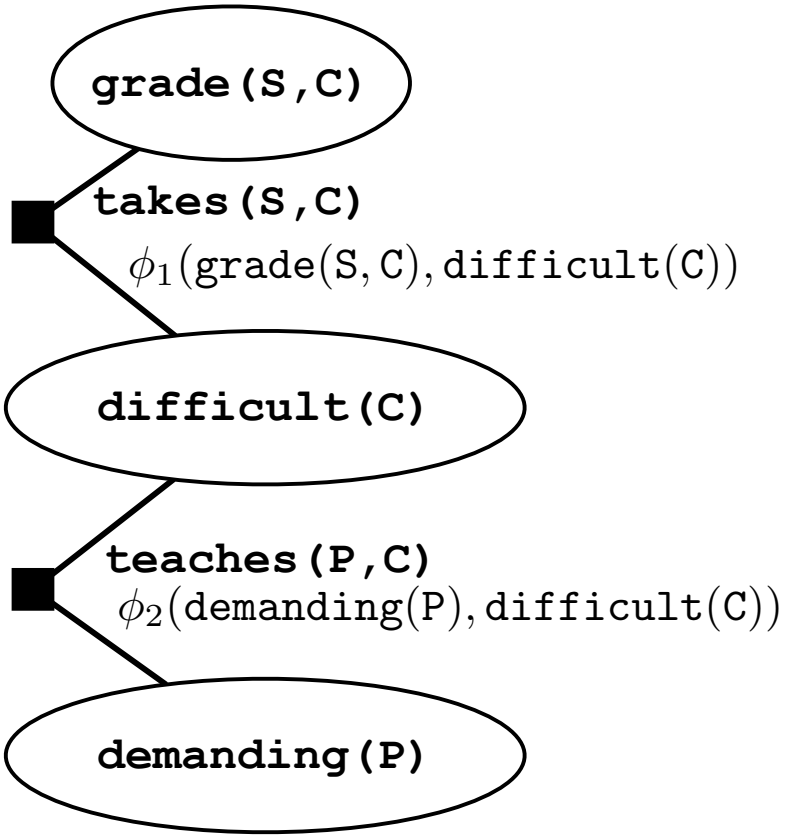


demanding(tom)?

1. construct factor graph
2. run any propositional inference technique

```
 takes(ann,ml).
 takes(bob,ml).
 takes(ann,db).
teaches(dan,db).
teaches(tom,ml).
```

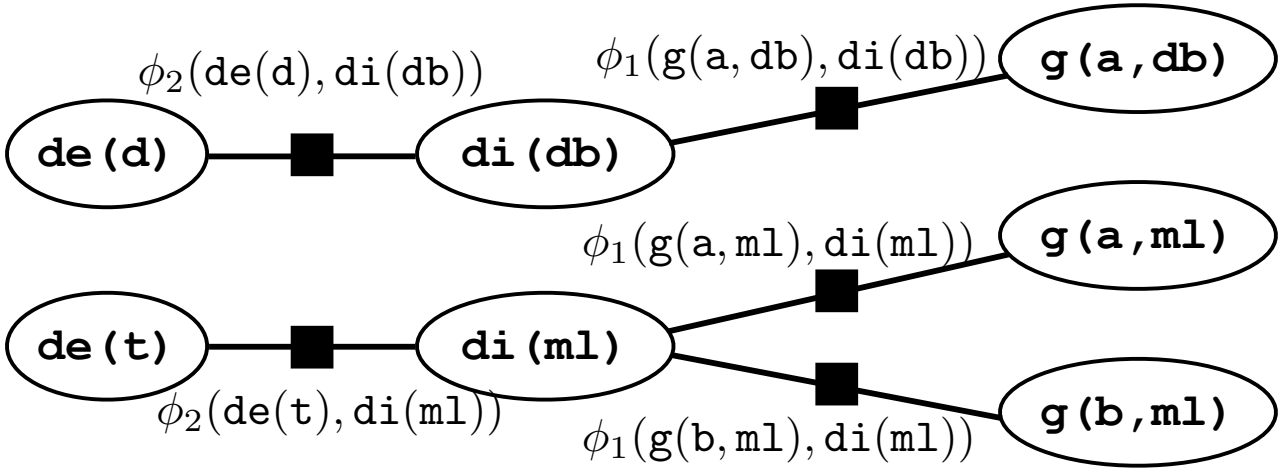
Inference by Grounding



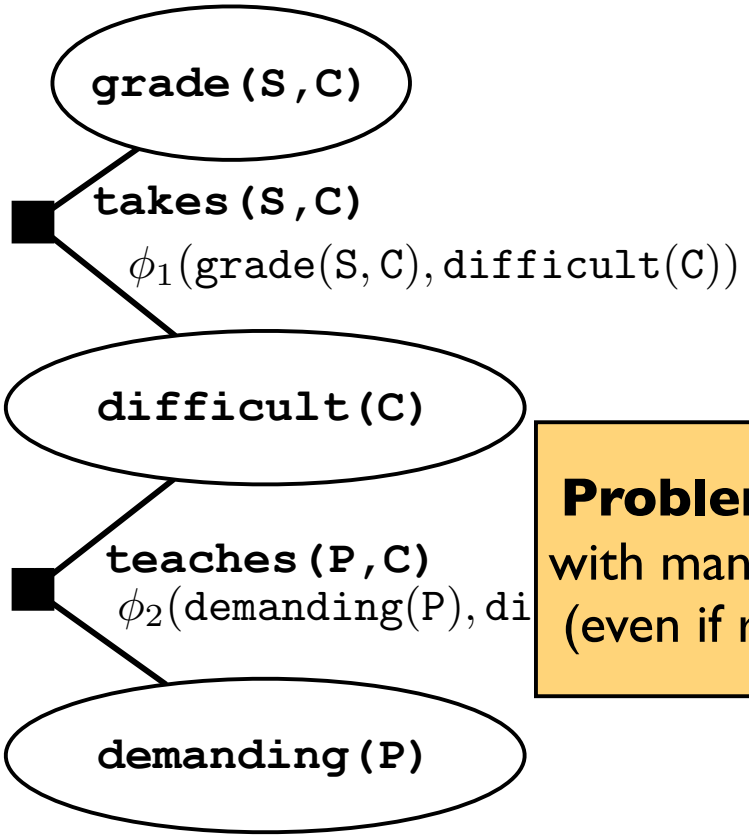
demanding(tom)?

1. construct factor graph
2. run any propositional inference technique

```
takes(ann,ml).
takes(bob,ml).
takes(ann,db).
teaches(dan,db).
teaches(tom,ml).
```



Inference by Grounding



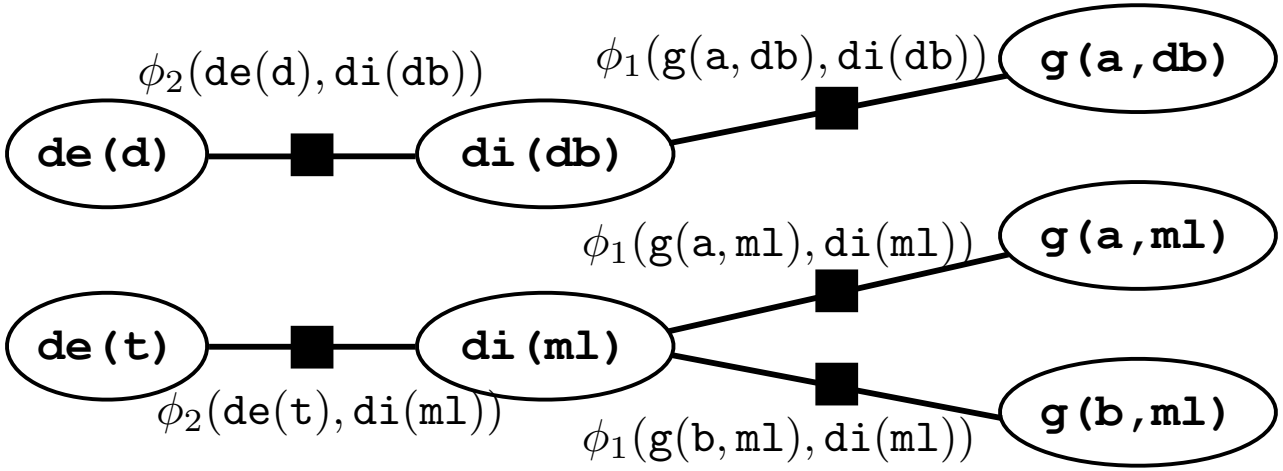
Problem: often huge factor graphs with many repetitions or symmetries (even if restricted to relevant parts)

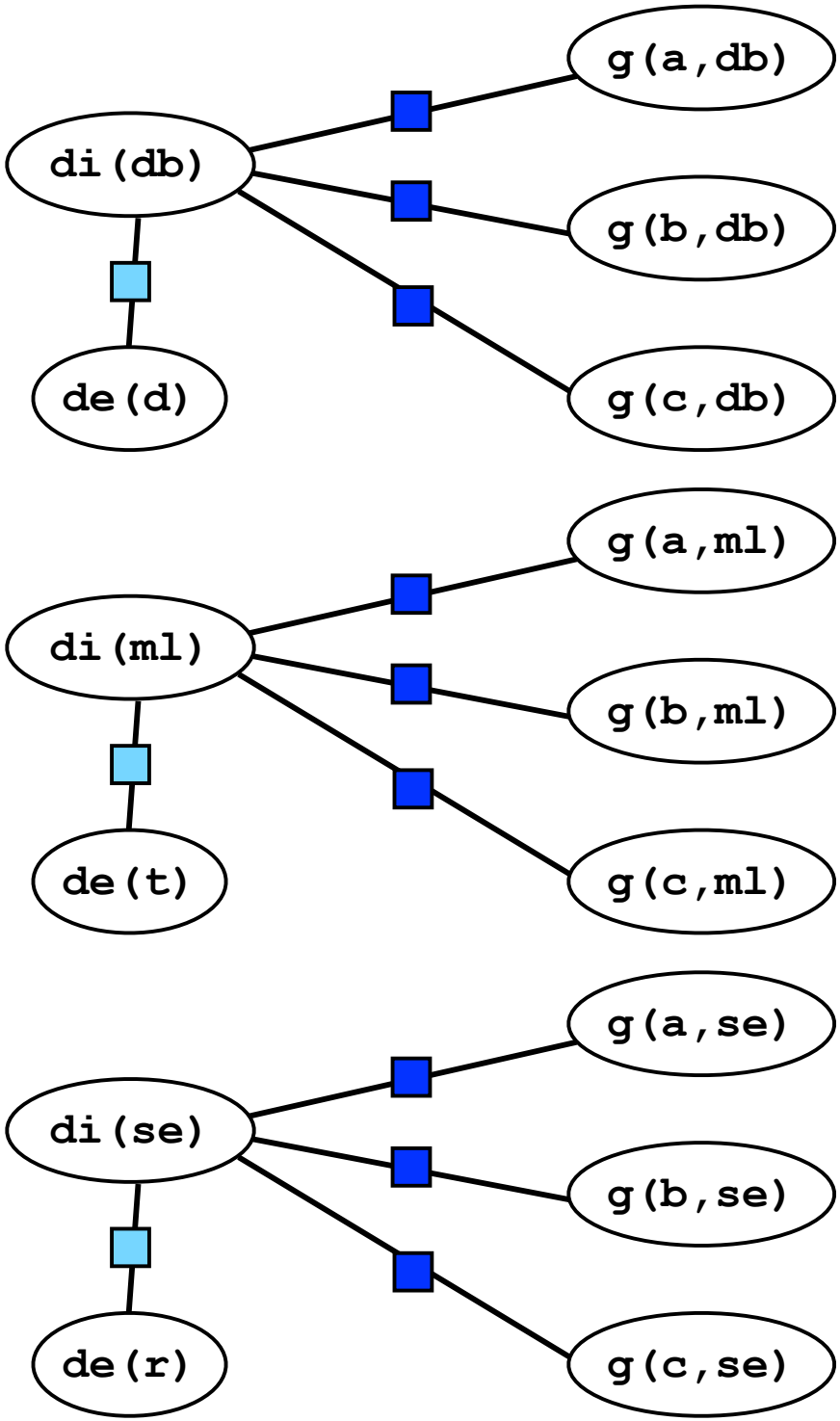
demanding(tom)?

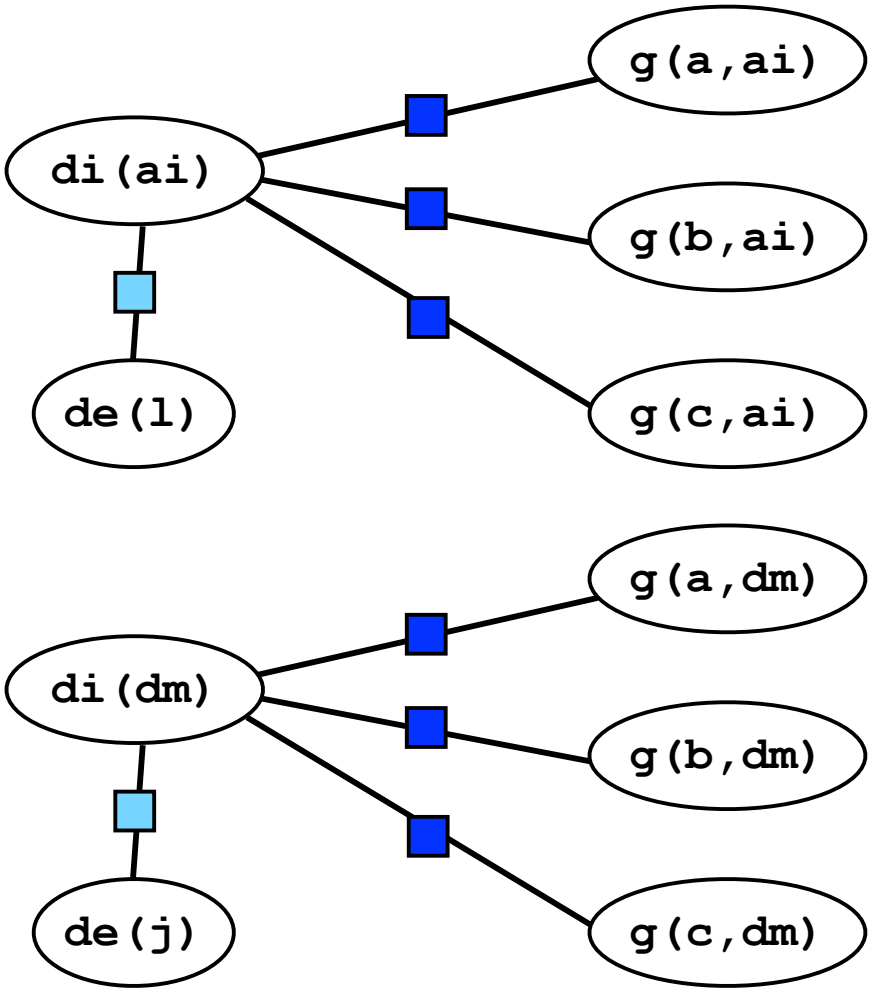
ctor graph

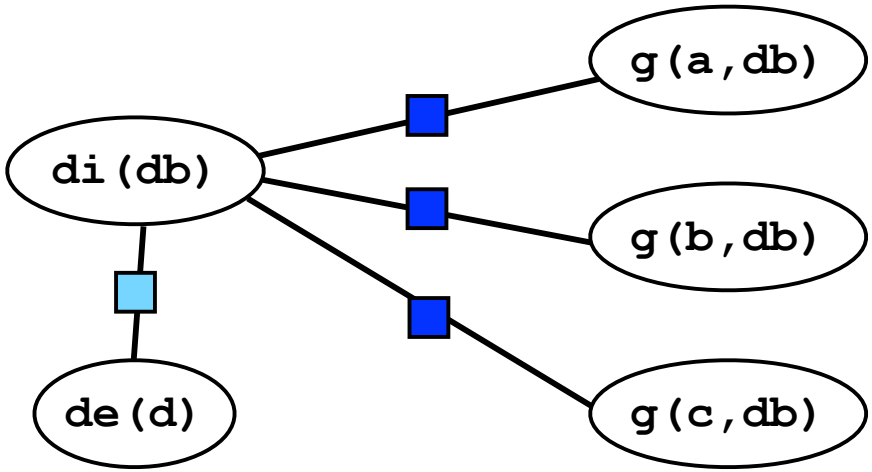
2. Run any propositional inference technique

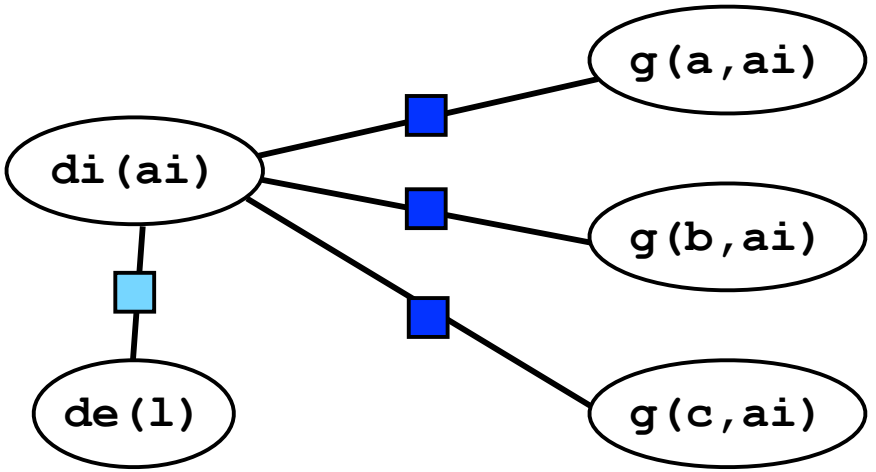
```
takes(ann,ml).
takes(bob,ml).
takes(ann,db).
teaches(dan,db).
teaches(tom,ml).
```

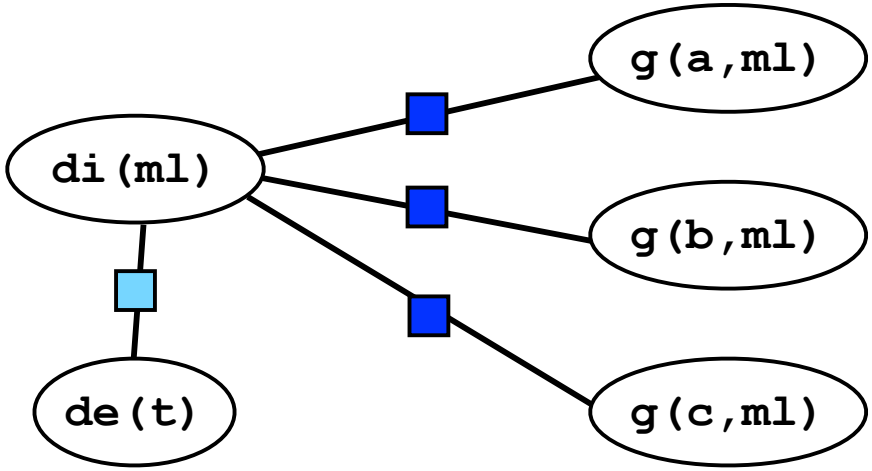


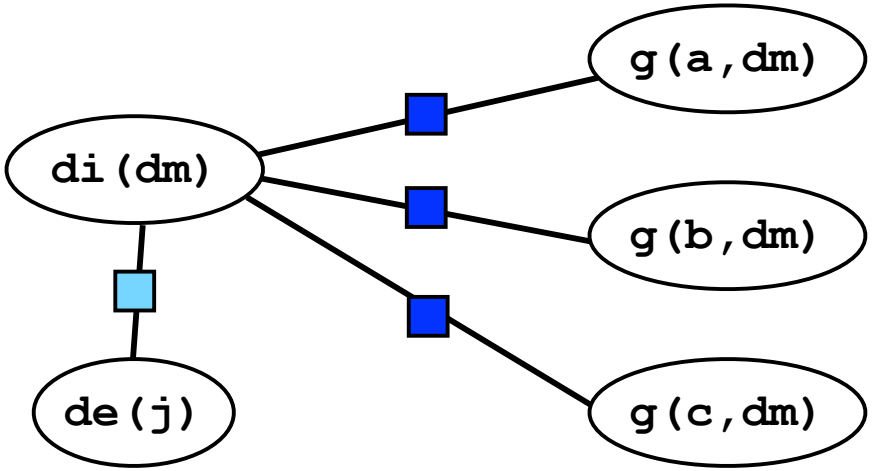

g(a,db)
di(db)
g(b,db)
de(d)
g(c,db)
g(a,ml)
di(ml)
g(b,ml)
de(t)
g(c,ml)
g(a,se)
di(se)
g(b,se)
de(r)
g(c,se)

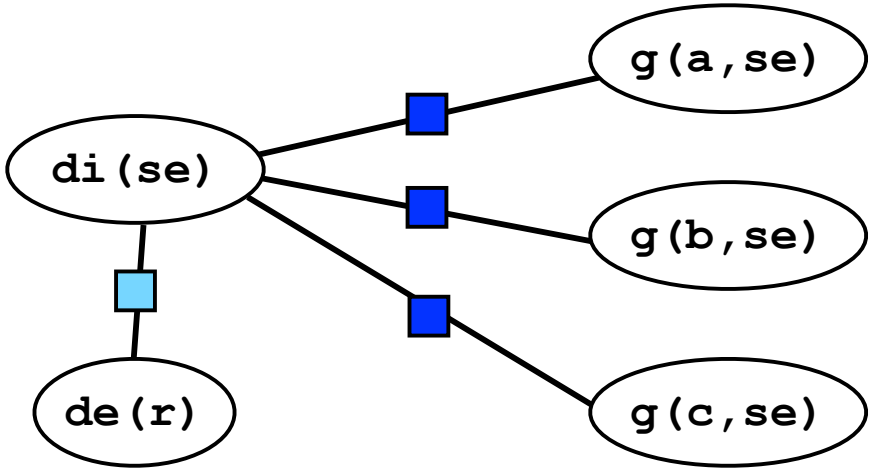

g(a,ai)
di(ai)
g(b,ai)
de(l)
g(c,ai)
g(a,dm)
di(dm)
g(b,dm)
de(j)
g(c,dm)



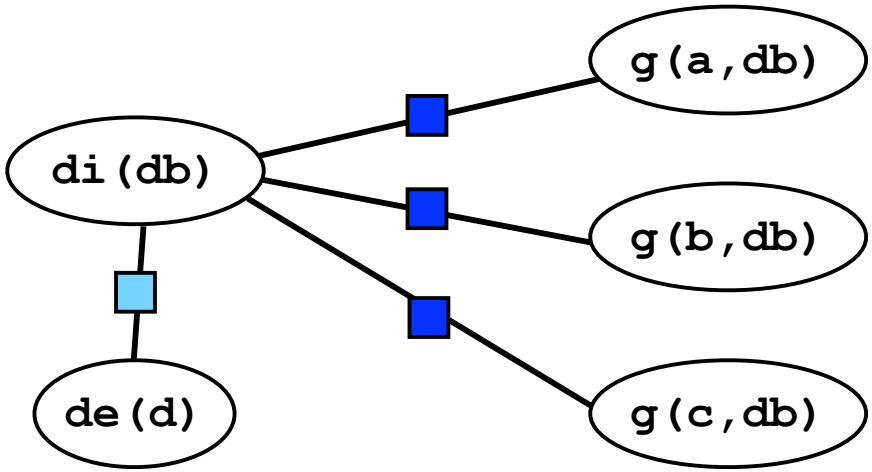


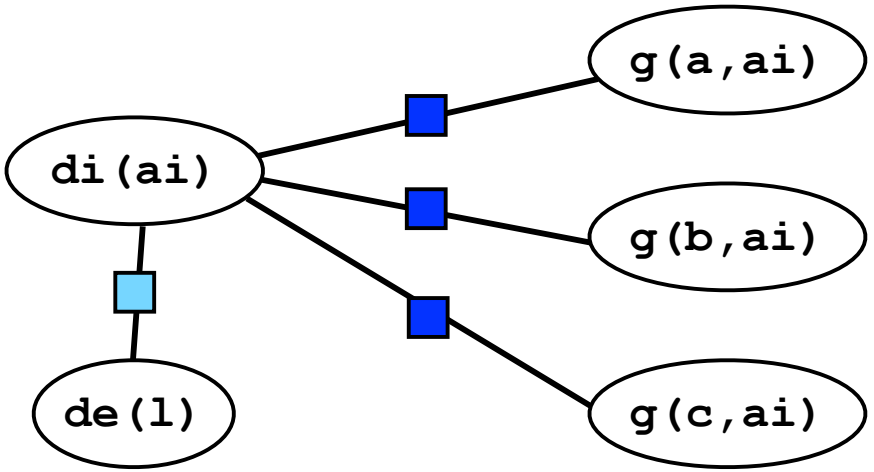


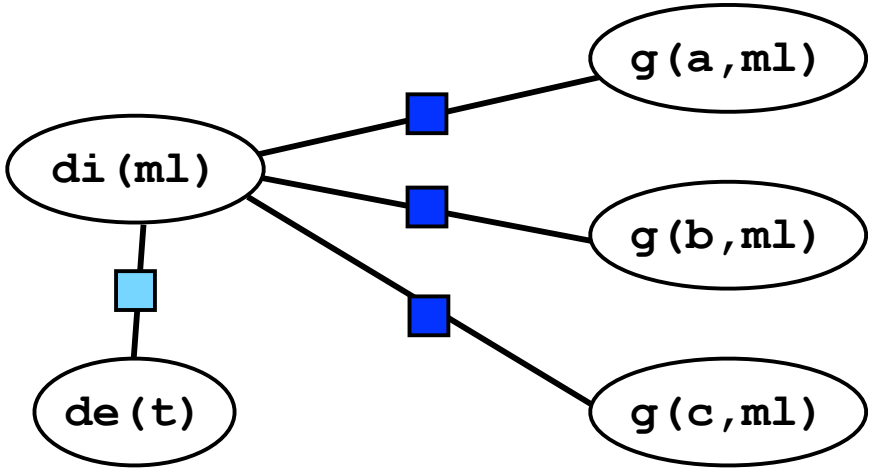


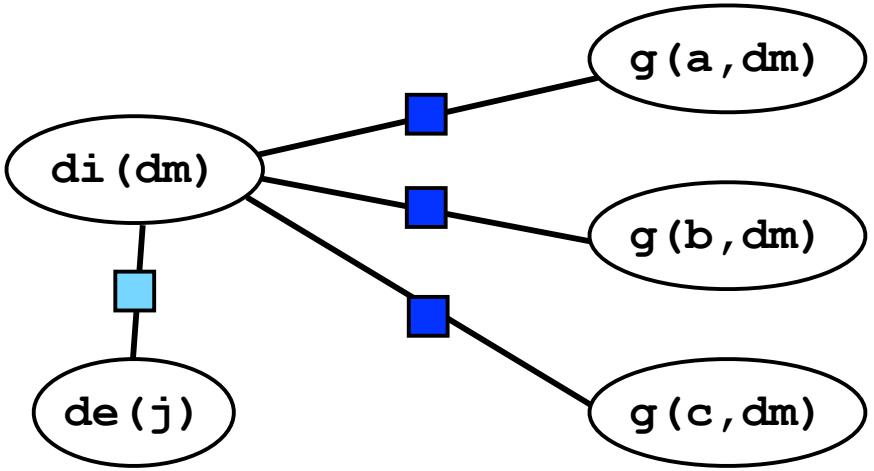


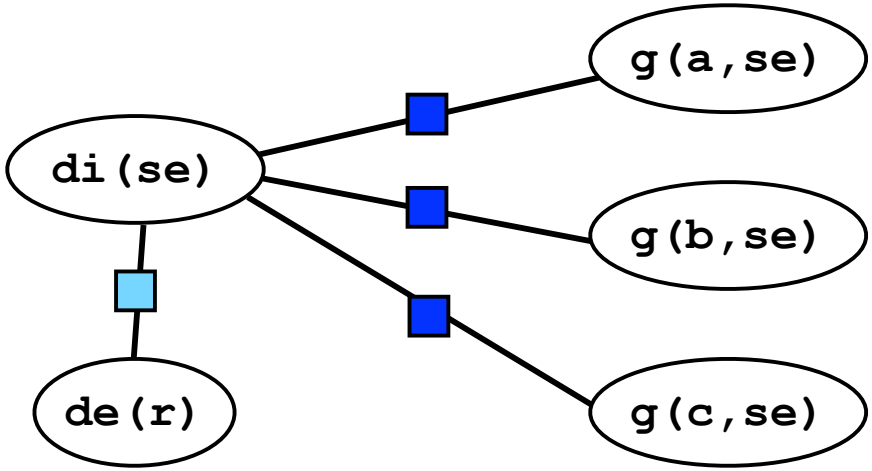
compute probability distribution over grades for every student-course pair





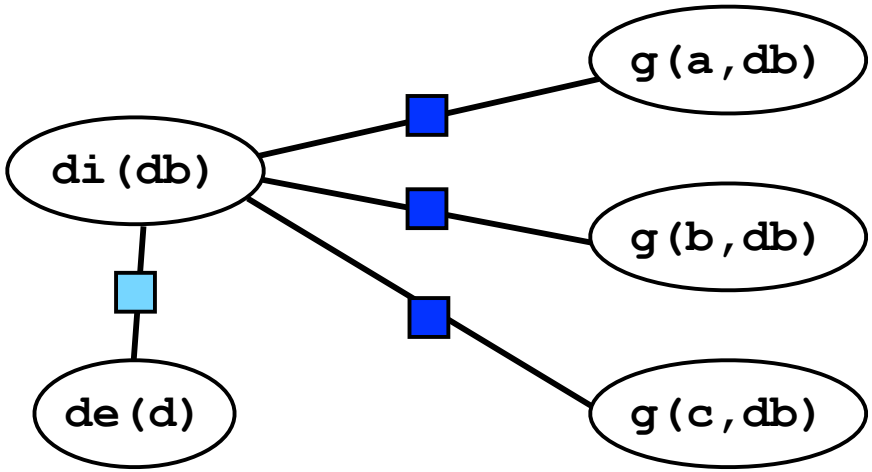


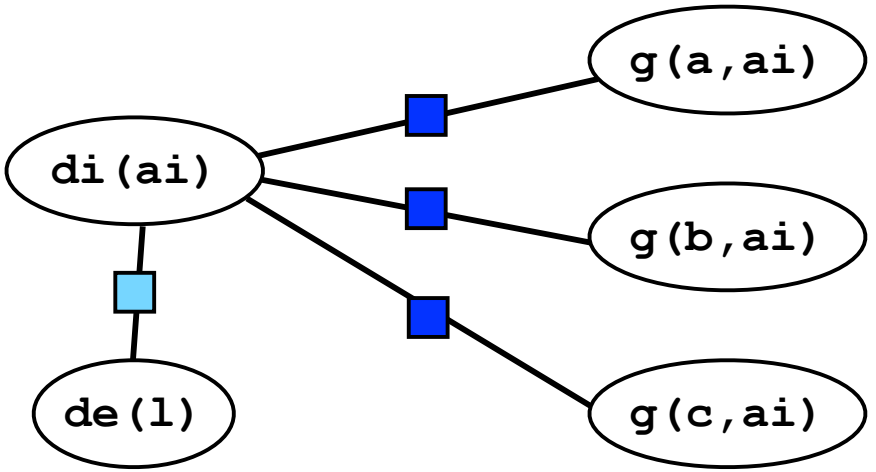


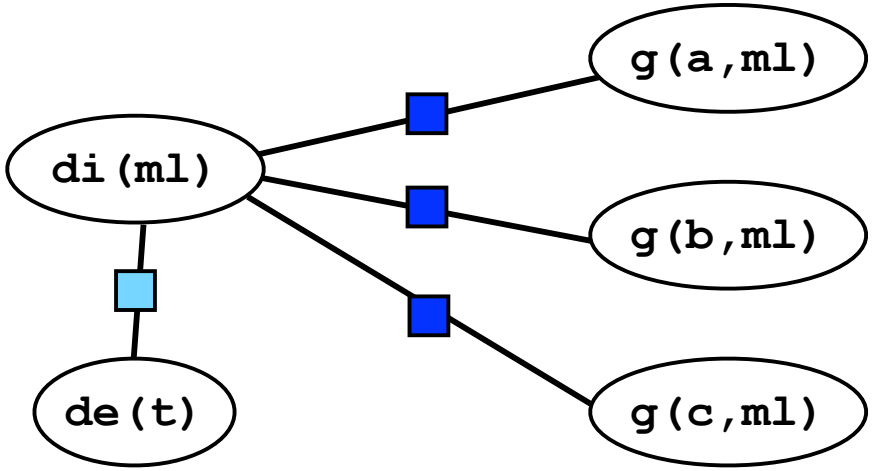


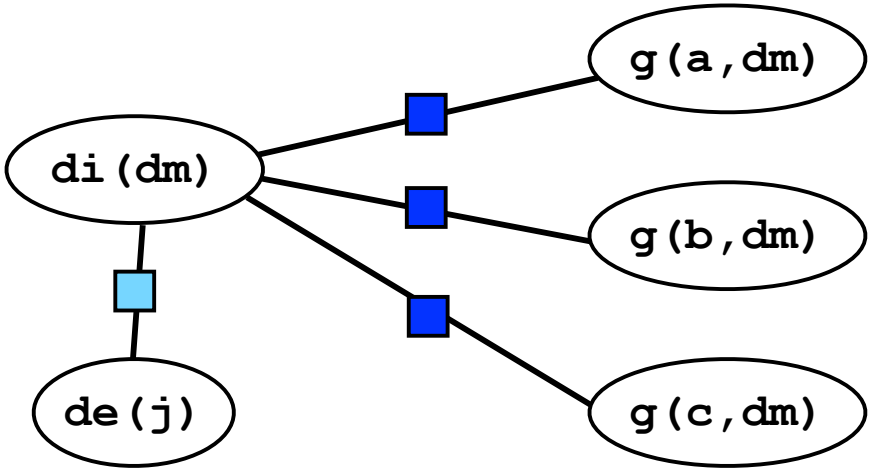
compute probability distribution over grades for every student-course pair

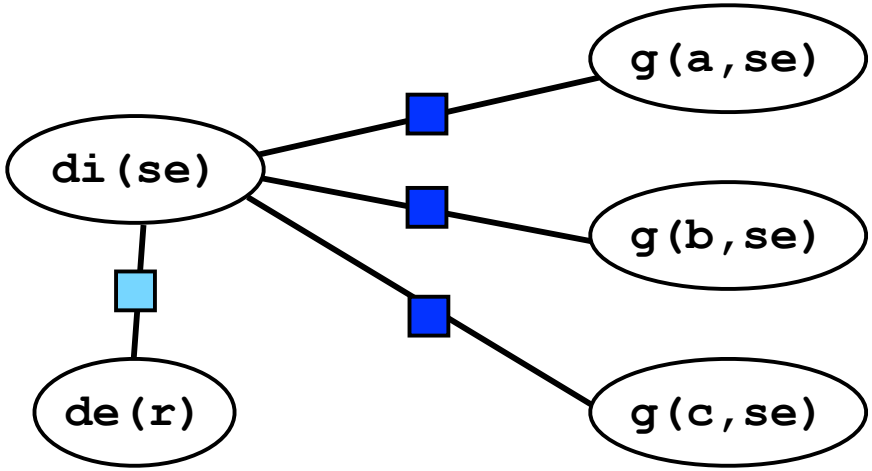
same computation for each pair!



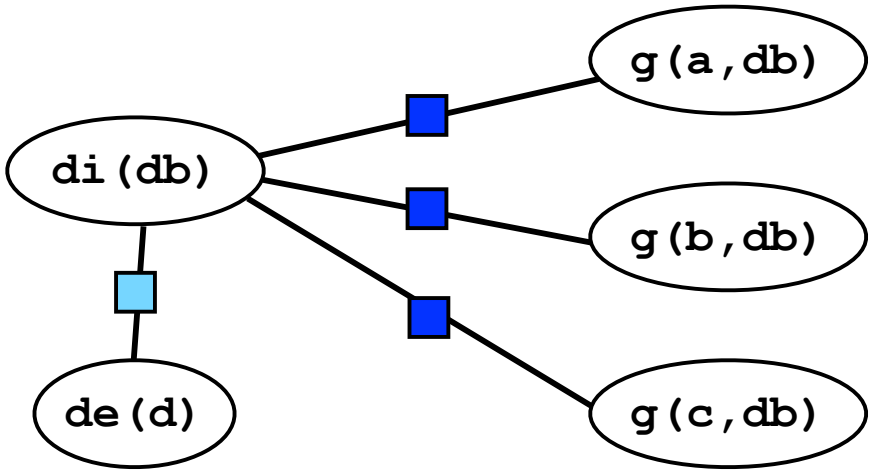


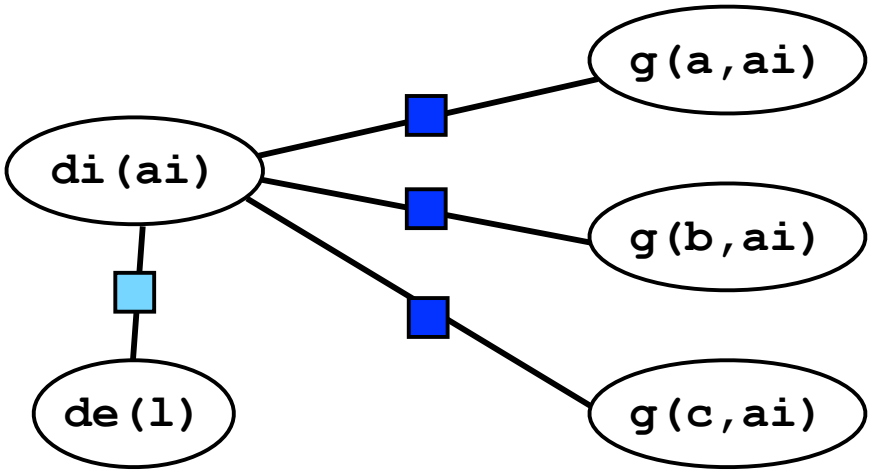


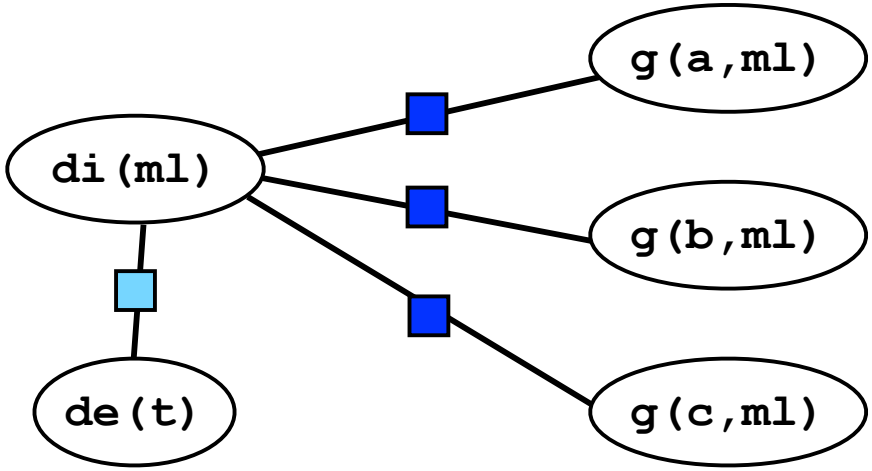


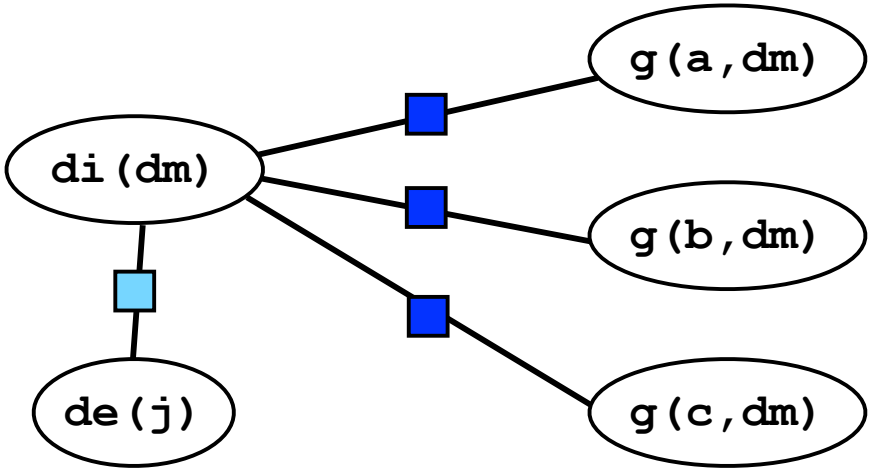


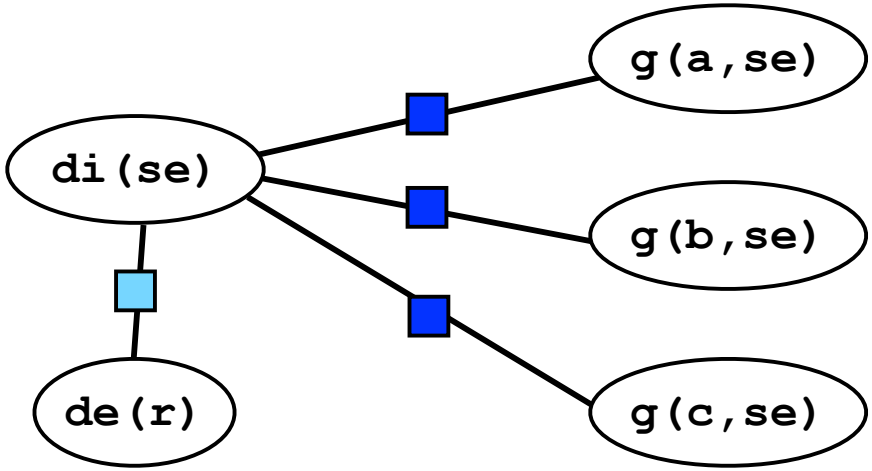
what is the probability that some professor is demanding?





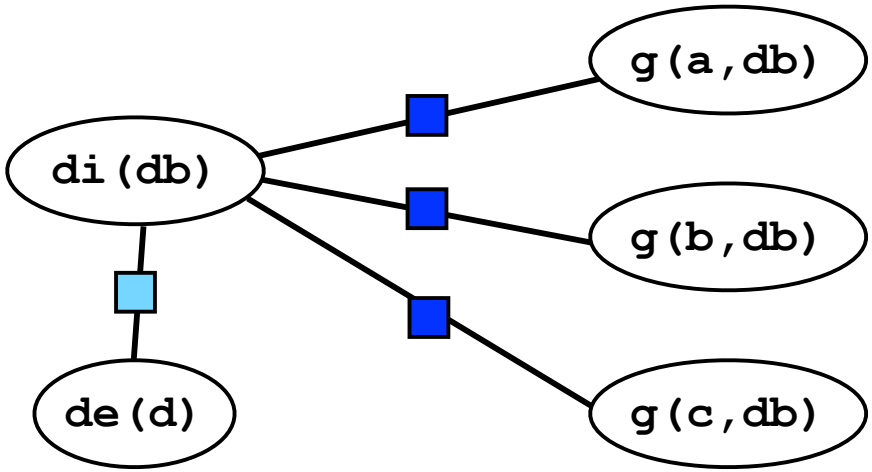


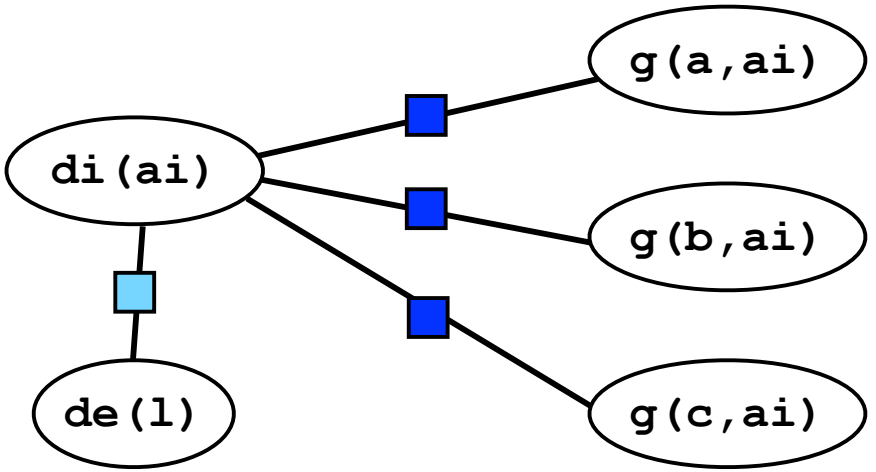


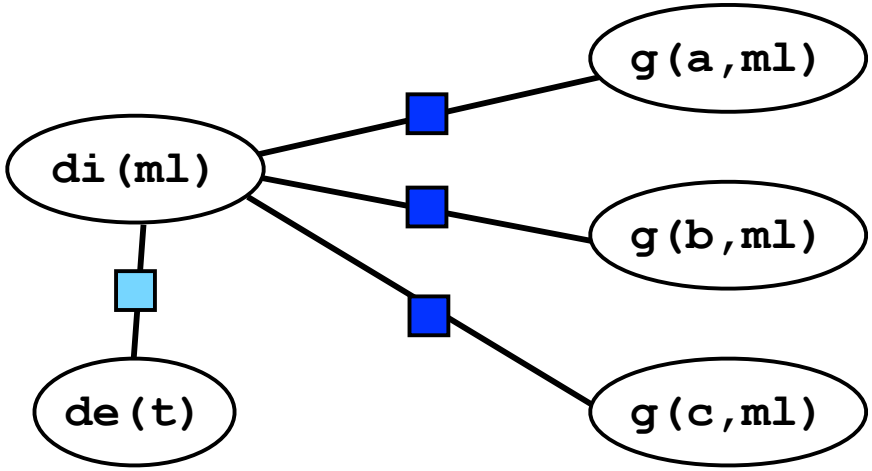


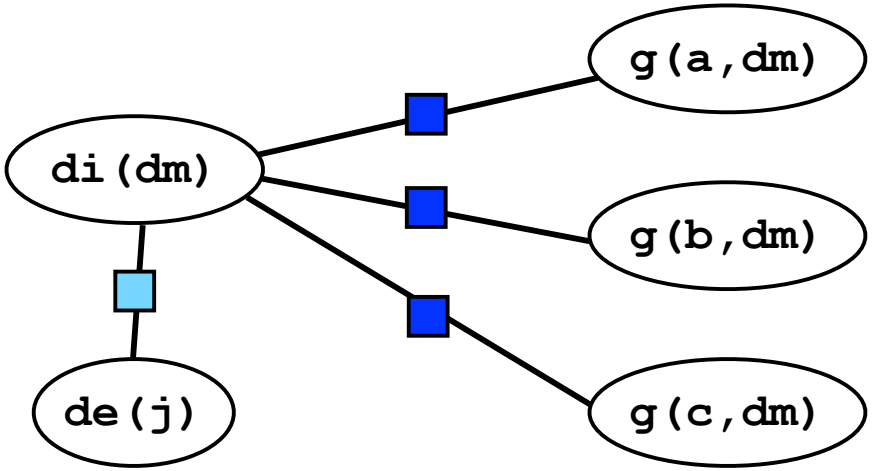
what is the probability that some professor is demanding?

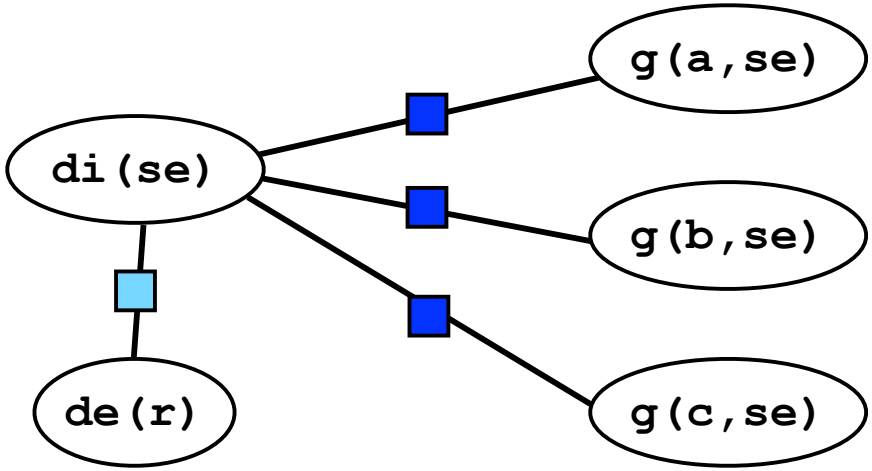
same computation for each professor!







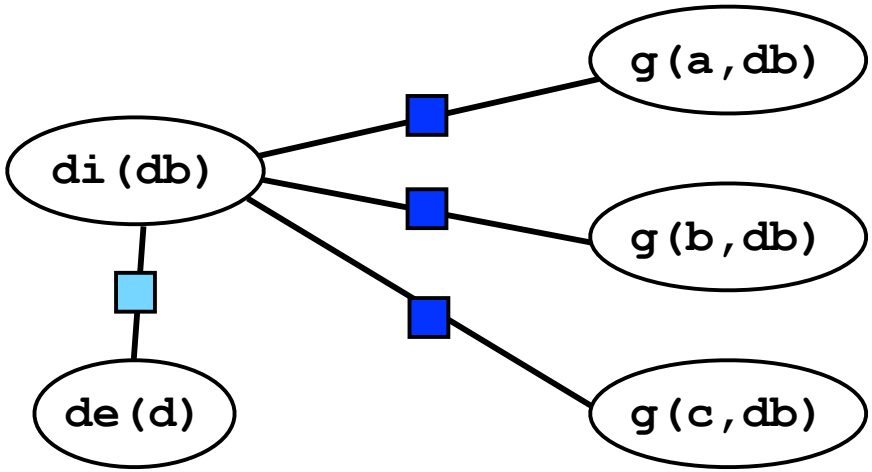


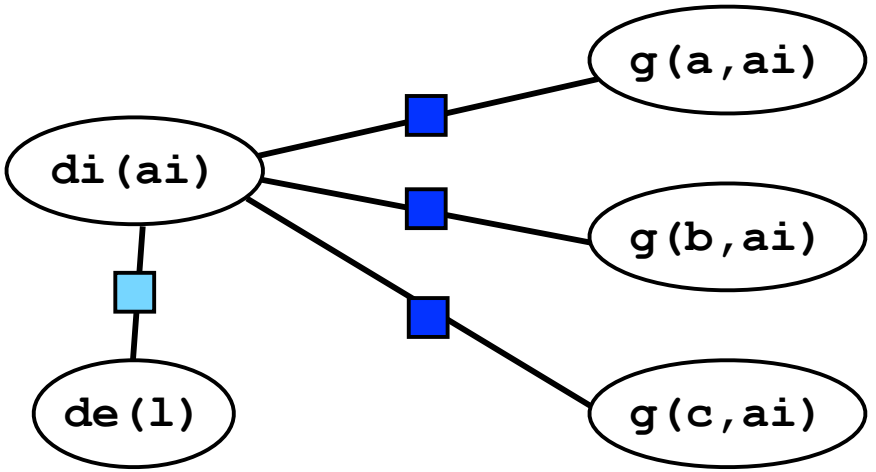


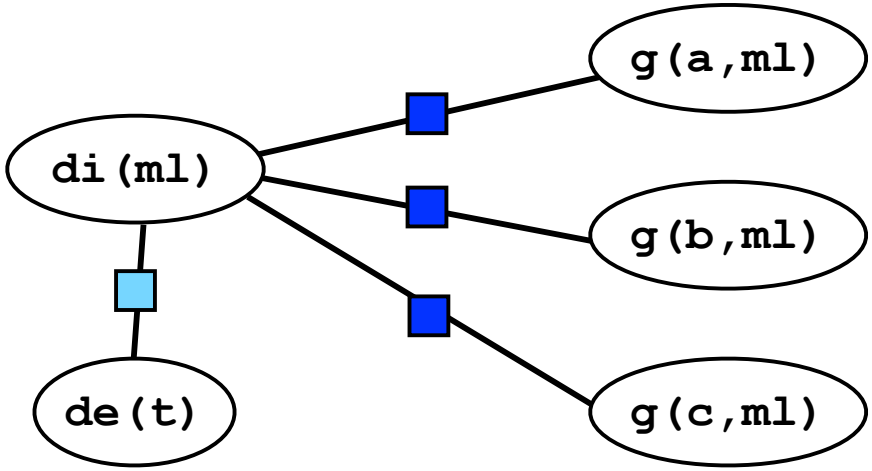
what is the probability that some professor is demanding?

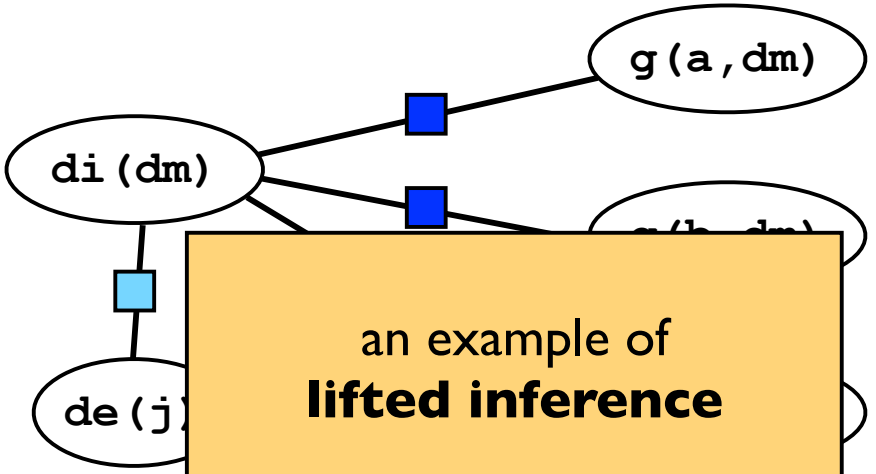
$$1 - \prod_{\mathtt{P}} (1 - P(\mathtt{de(P)})) = 1 - (1 - P(\mathtt{de(someP)}))^{\#\mathtt{P}}$$

$$= 1 - (1 - P(\mathtt{de(r)}))^5$$

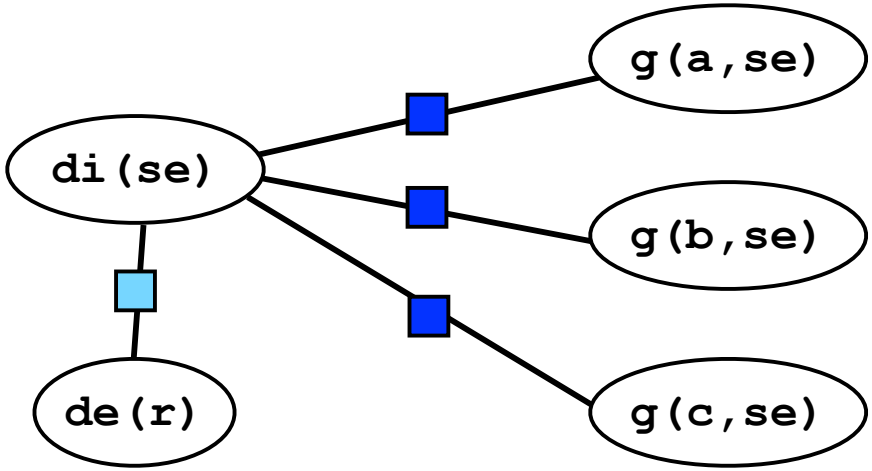








an example of
lifted inference

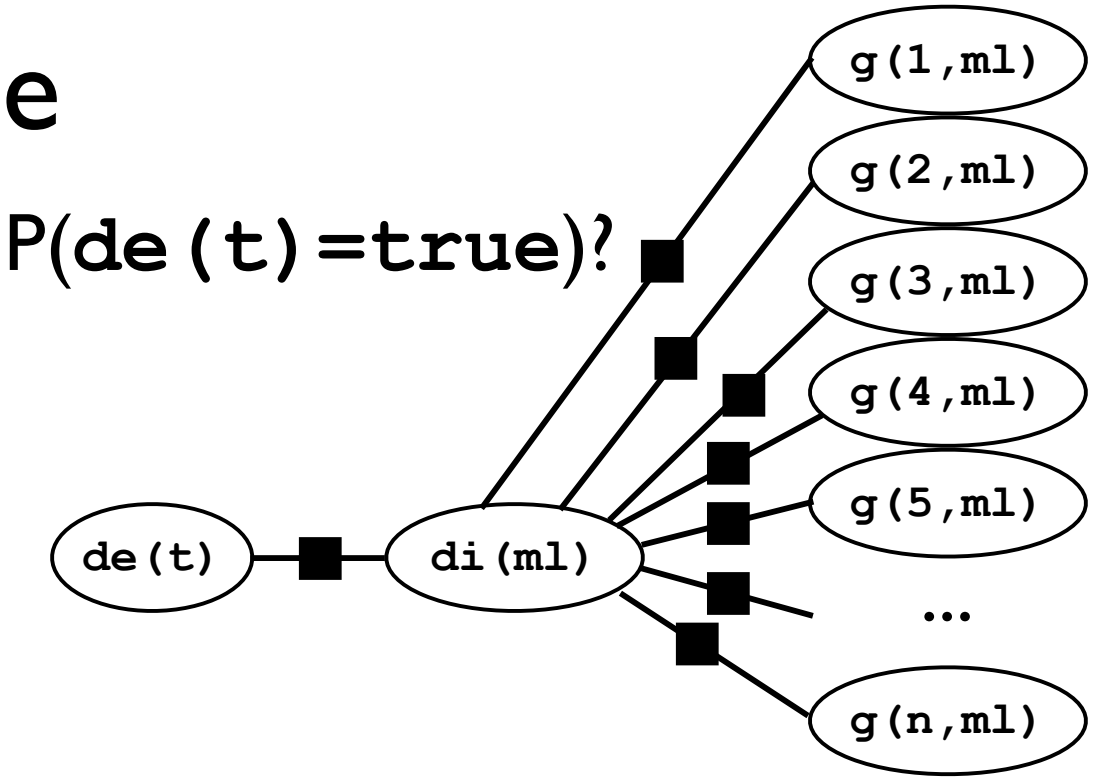


what is the probability that some professor is demanding?

$$1 - \prod_{\mathtt{P}} (1 - P(\mathtt{de(P)})) = 1 - (1 - P(\mathtt{de(someP)}))^{\#\mathtt{P}}$$

$$= 1 - (1 - P(\mathtt{de(r)}))^5$$

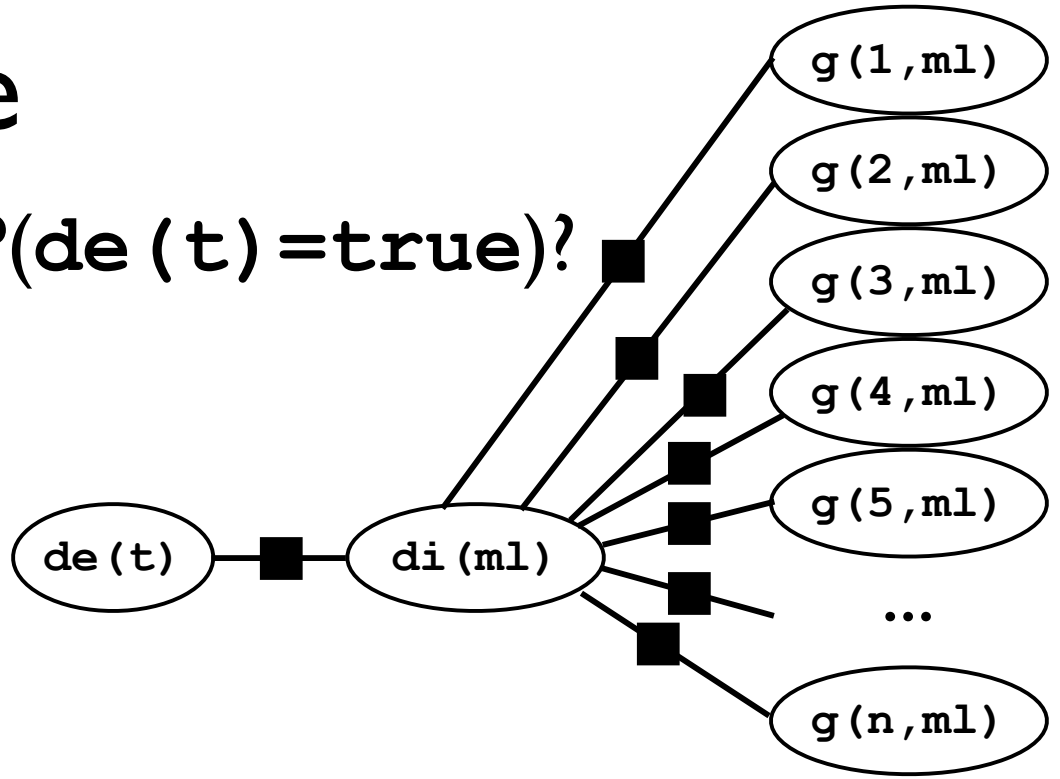
Another example



Another example

sum $\phi_2(\text{true}, d) \prod_{i=1..n} \phi_1(g_i, d)$

for all combinations of difficulty d and students' grades (g_1,...,g_n)

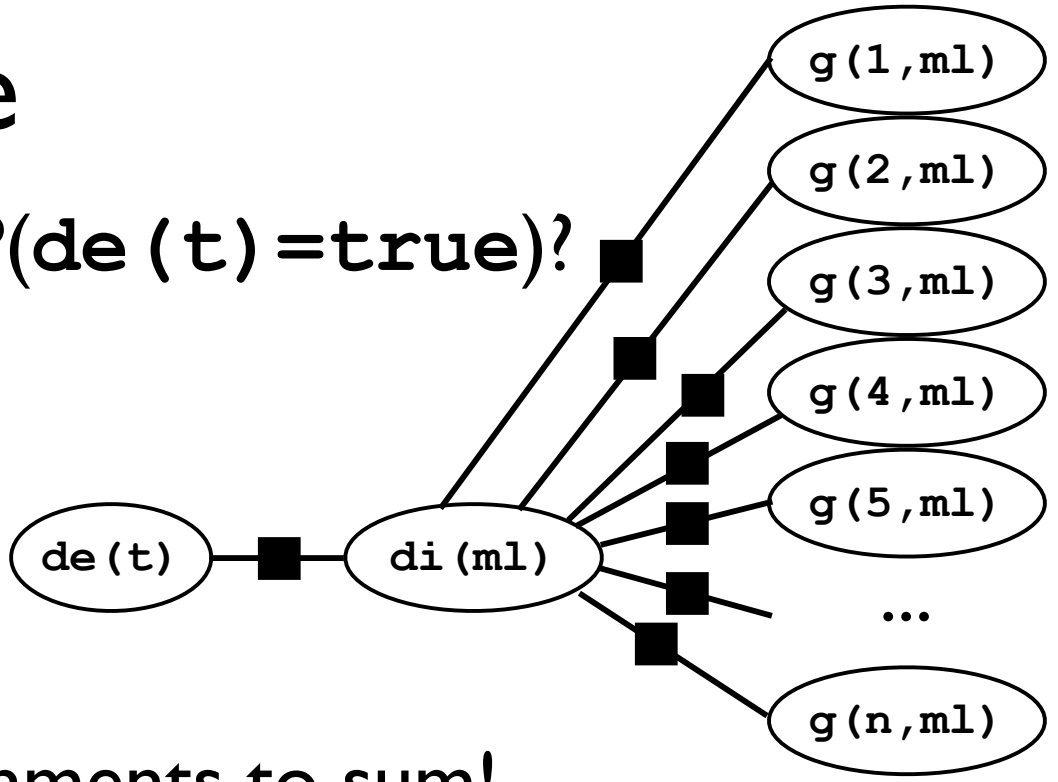


Another example

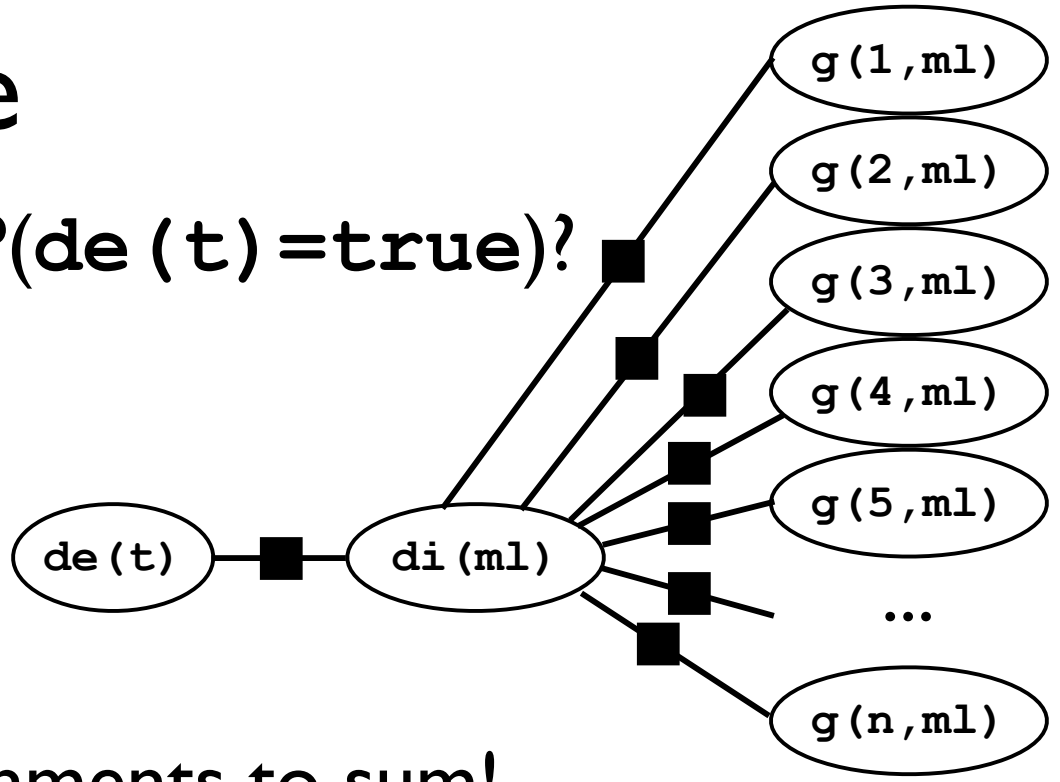
sum $\phi_2(\text{true}, d) \prod_{i=1..n} \phi_1(g_i, d)$

for all combinations of difficulty d and students' grades (g_1,...,g_n)

O(#grades$^{\#\text{students}}$) assignments to sum!



Another example



sum $\phi_2(\text{true}, d) \prod_{i=1..n} \phi_1(g_i, d)$

for all combinations of difficulty d and students' grades (g_1,...,g_n)

O(#grades$^{\#\text{students}}$) assignments to sum!

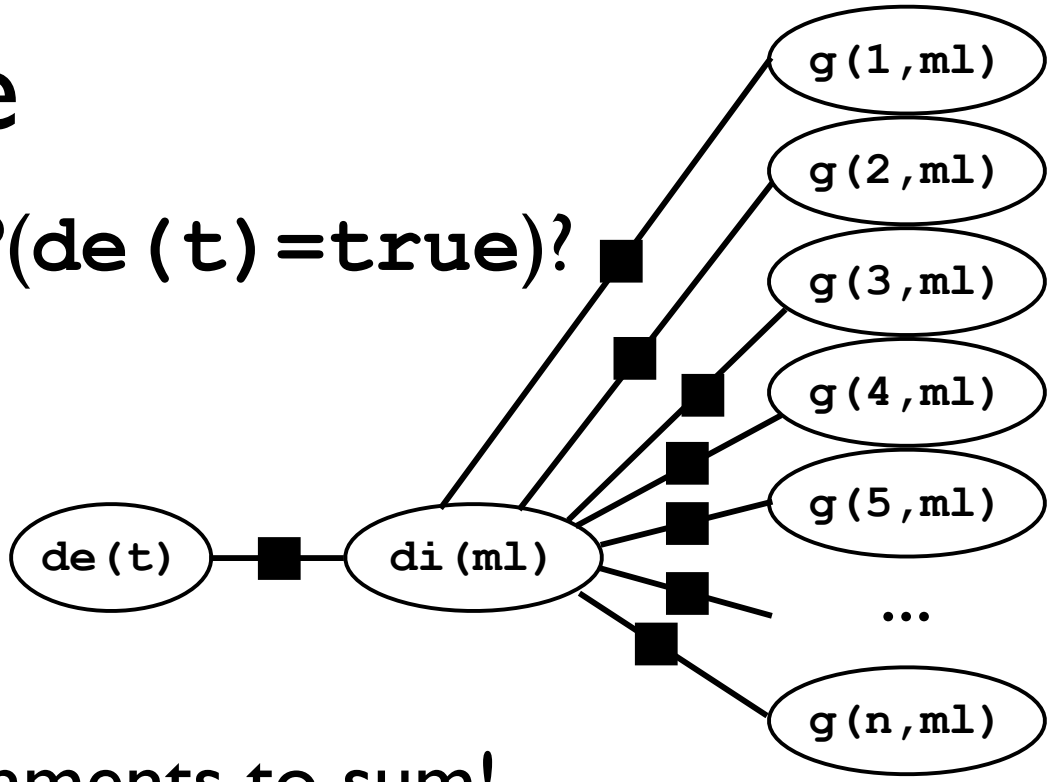
but: identity of students doesn't matter!

sufficient to count how often each grade m_1,...,m_k appears

Another example

sum $\phi_2(\text{true}, d) \prod_{i=1..n} \phi_1(g_i, d)$

for all combinations of difficulty d and students' grades (g_1,...,g_n)



O(#grades$^{\#\text{students}}$) assignments to sum!

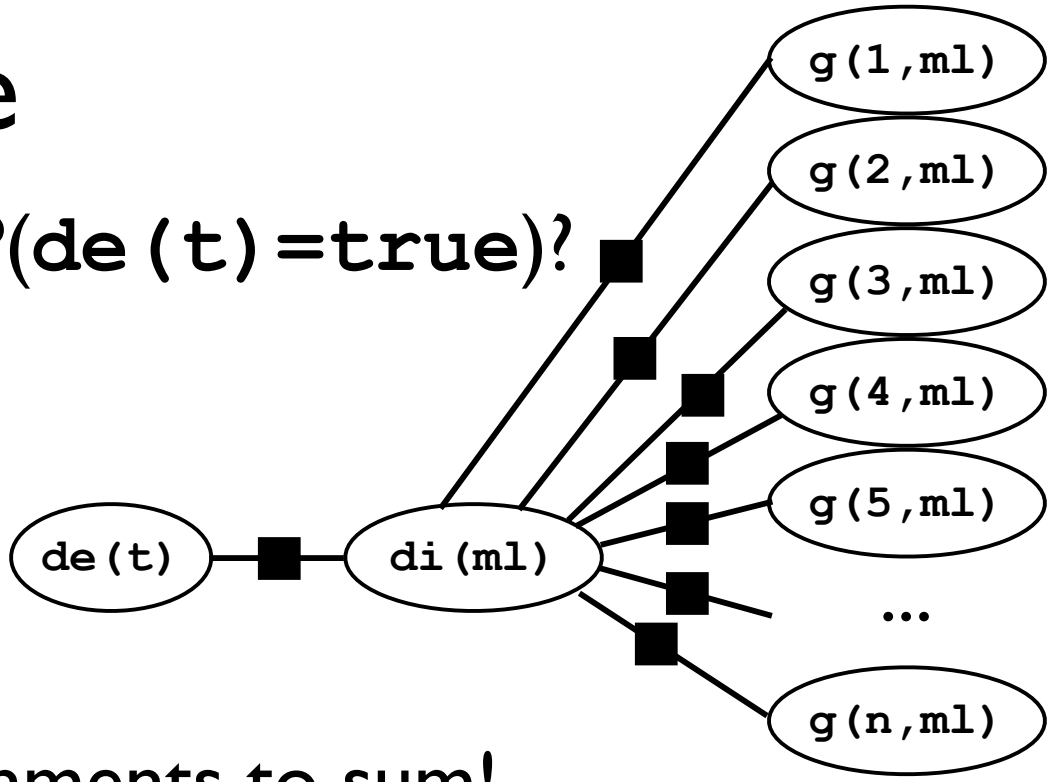
but: identity of students doesn't matter!
sufficient to count how often each grade m_1,...,m_k appears

sum $\phi_2(\text{true}, d) \prod_{i=1..k} \phi_1(m_i, d)^{\#m_i}$ instead

Another example

sum $\phi_2(\text{true}, d) \prod_{i=1..n} \phi_1(g_i, d)$

for all combinations of difficulty d and students' grades (g_1,...,g_n)



O(#grades$^{\#\text{students}}$) assignments to sum!

but: identity of students doesn't matter!
sufficient to count how often each grade m_1,...,m_k appears

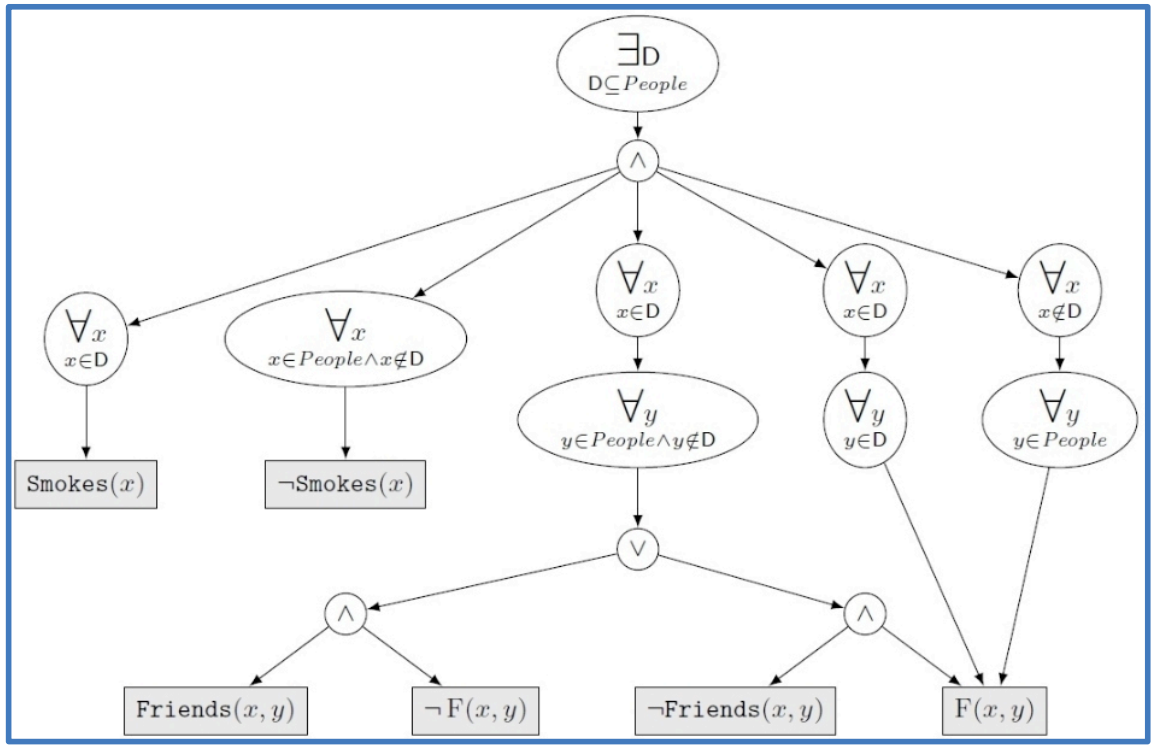
sum $\phi_2(\text{true}, d) \prod_{i=1..k} \phi_1(m_i, d)^{\#m_i}$ instead

e.g., k=3, n=15: >14M grade vectors vs 136 count vectors

Lifted Inference

- exploiting symmetries & repeated structure
- reasoning on first order level as much as possible
- aiming at independence from number of objects
- approximation: grouping similar computations
- very active research area

Example: Weighted First-Order Model Counting (WFOMC)



First-Order d-DNNF Circuit

Smokes → 1
¬Smokes → 1
Friends → 1
¬Friends → 1
F → exp(3.14)
¬F → 1

Weight Function

Alice
Bob
Charlie

Domain

Weighted First-Order Model Count is 1479.85

Slide Guy Van den Broeck

Roadmap

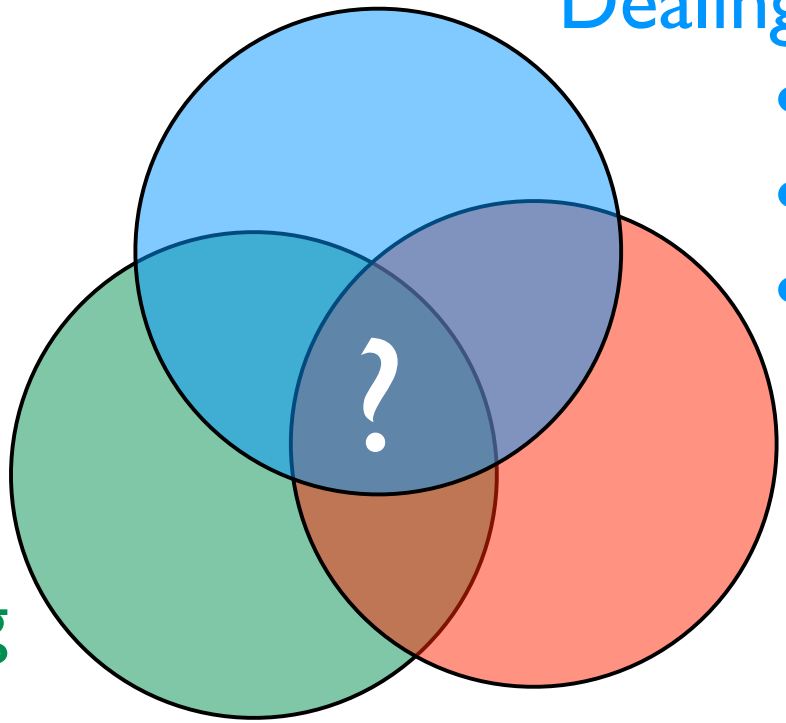
- Modeling
- Inference
- Learning & Dynamics
- Advanced Inference and KBMC

... with some detours on the way

A key question in AI:

Dealing with uncertainty
- probability theory
- graphical models
- ...

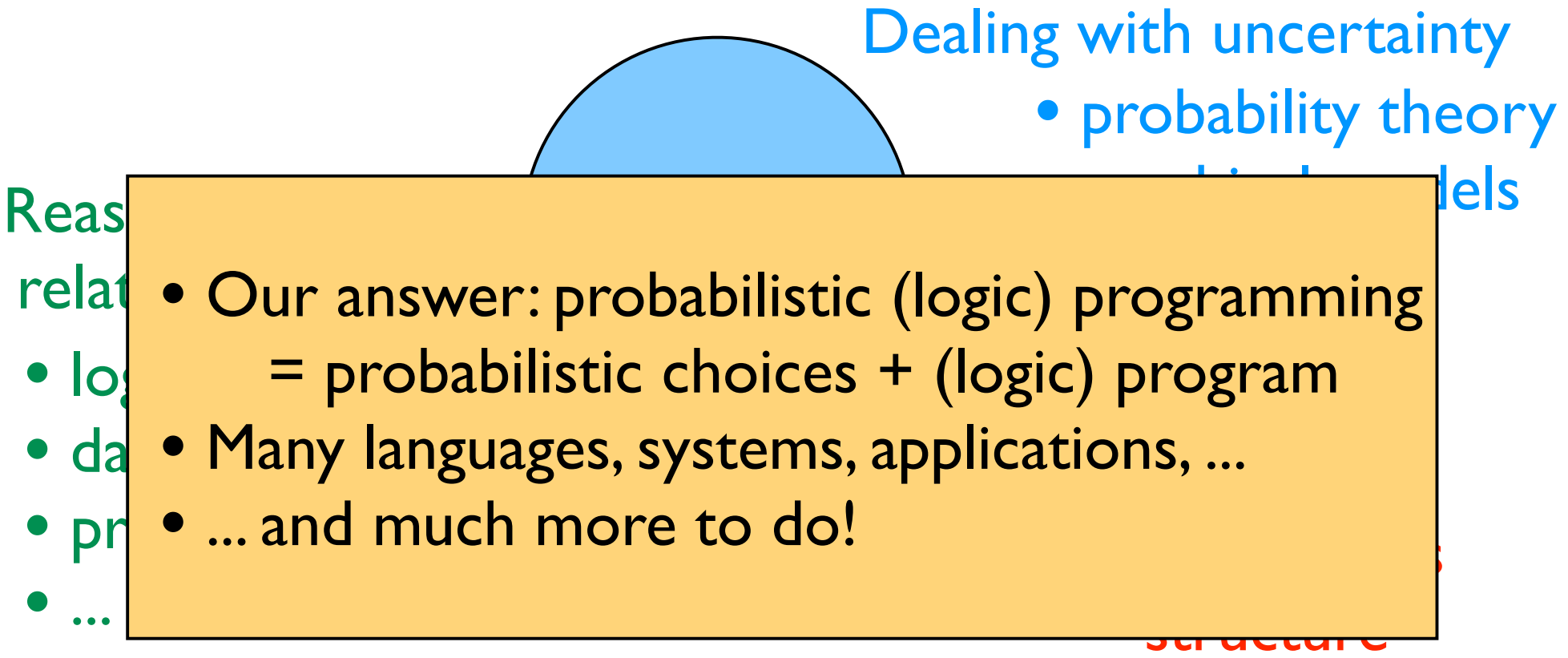
Reasoning with relational data
- logic
- databases
- programming
- ...



Learning
- parameters
- structure

Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

A key question in AI:

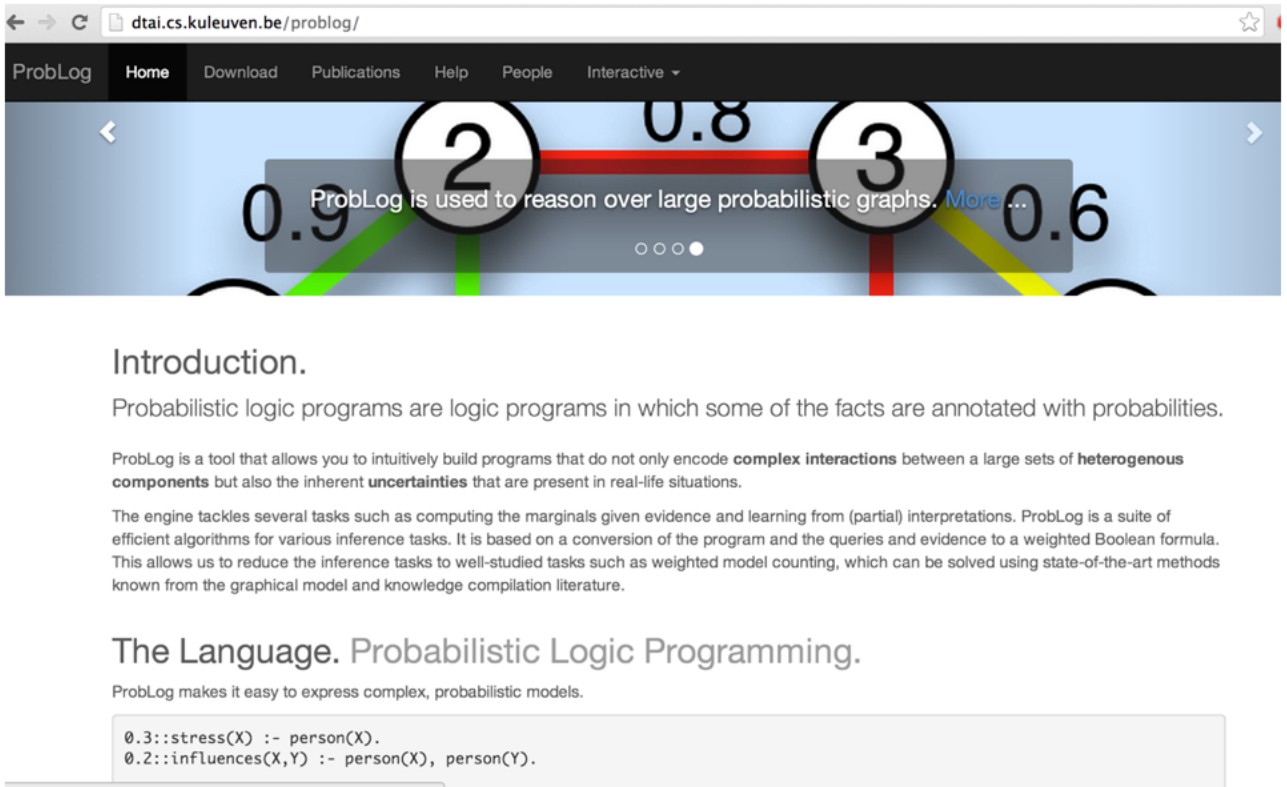


Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

Maurice Bruynooghe
Bart Demoen
Anton Dries
Daan Fierens
Jason Filippou
Bernd Gutmann
Manfred Jaeger
Gerda Janssens
Kristian Kersting
Theofrastos Mantadelis
Wannes Meert
Bogdan Moldovan
Siegfried Nijssen
Davide Nitti
Joris Renkens
Kate Revoredo
Ricardo Rocha
Vitor Santos Costa
Dimitar Shterionov
Ingo Thon
Hannu Toivonen
Guy Van den Broeck
Mathias Verbeke
Jonas Vlasselaer

Thanks !

http://dtai.cs.kuleuven.be/problog



PLP Systems

- **PRISM** http://sato-www.cs.titech.ac.jp/prism/
- **ProbLog2** http://dtai.cs.kuleuven.be/problog/
- Yap Prolog http://www.dcc.fc.up.pt/~vsc/Yap/ includes
  - **ProbLog1**
  - **cplint** https://sites.google.com/a/unife.it/ml/cplint
  - **CLP(BN)**
  - **LP2**
- **PITA** in XSB Prolog http://xsb.sourceforge.net/
- **AILog2** http://artint.info/code/ailog/ailog2.html
- **SLPs** http://stoics.org.uk/~nicos/sware/pepl
- **contdist** http://www.cs.sunysb.edu/~cram/contdist/
- **DC** https://code.google.com/p/distributional-clauses
- **WFOMC** http://dtai.cs.kuleuven.be/ml/systems/wfomc

References

Bach SH, Broecheler M, Getoor L, O'Leary DP (2012) Scaling MPE inference for constrained continuous Markov random fields with consensus optimization. In: Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS-12)

Broecheler M, Mihalkova L, Getoor L (2010) Probabilistic similarity logic. In: Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI-10)

Bryant RE (1986) Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers 35(8):677–691

Cohen SB, Simmons RJ, Smith NA (2008) Dynamic programming algorithms as products of weighted logic programs. In: Proceedings of the 24th International Conference on Logic Programming (ICLP-08)

Cussens J (2001) Parameter estimation in stochastic logic programs. Machine Learning 44(3):245–271

De Maeyer D, Renkens J, Cloots L, De Raedt L, Marchal K (2013) Phenetic: network-based interpretation of unstructured gene lists in e. coli. Molecular BioSystems 9(7):1594–1603

De Raedt L, Kimmig A (2013) Probabilistic programming concepts. CoRR abs/1312.4328

De Raedt L, Kimmig A, Toivonen H (2007) ProbLog: A probabilistic Prolog and its application in link discovery. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)

De Raedt L, Frasconi P, Kersting K, Muggleton S (eds) (2008) Probabilistic Inductive Logic Programming — Theory and Applications, Lecture Notes in Artificial Intelligence, vol 4911. Springer

Eisner J, Goldlust E, Smith N (2005) Compiling Comp Ling: Weighted dynamic programming and the Dyna language. In: Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP-05)

Fierens D, Blockeel H, Bruynooghe M, Ramon J (2005) Logical Bayesian networks and their relation to other probabilistic logical models. In: Proceedings of the 15th International Conference on Inductive Logic Programming (ILP-05)

Fierens D, Van den Broeck G, Bruynooghe M, De Raedt L (2012) Constraints for probabilistic logic programming. In: Proceedings of the NIPS Probabilistic Programming Workshop

Fierens D, Van den Broeck G, Renkens J, Shterionov D, Gutmann B, Thon I, Janssens G, De Raedt L (2014) Inference and learning in probabilistic logic programs using weighted Boolean formulas. Theory and Practice of Logic Programming (TPLP) FirstView

Getoor L, Friedman N, Koller D, Pfeffer A, Taskar B (2007) Probabilistic relational models. In: Getoor L, Taskar B (eds) An Introduction to Statistical Relational Learning, MIT Press, pp 129–174

Goodman N, Mansinghka VK, Roy DM, Bonawitz K, Tenenbaum JB (2008) Church: a language for generative models. In: Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI-08)

Gutmann B, Thon I, De Raedt L (2011a) Learning the parameters of probabilistic logic programs from interpretations. In: Proceedings of the 22nd European Conference on Machine Learning (ECML-11)

Gutmann B, Thon I, Kimmig A, Bruynooghe M, De Raedt L (2011b) The magic of logical inference in probabilistic programming. Theory and Practice of Logic Programming (TPLP) 11((4–5)):663–680

Huang B, Kimmig A, Getoor L, Golbeck J (2013) A flexible framework for probabilistic models of social trust. In: Proceedings of the International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction (SBP-13)

Jaeger M (2002) Relational Bayesian networks: A survey. Linköping Electronic Articles in Computer and Information Science 7(015)

Kersting K, Raedt LD (2001) Bayesian logic programs. CoRR cs.AI/0111058

Kimmig A, Van den Broeck G, De Raedt L (2011a) An algebraic Prolog for reasoning about possible worlds. In: Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-11)

Kimmig A, Demoen B, De Raedt L, Santos Costa V, Rocha R (2011b) On the implementation of the probabilistic logic programming language ProbLog. Theory and Practice of Logic Programming (TPLP) 11:235–262

Koller D, Pfeffer A (1998) Probabilistic frame-based systems. In: Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)

McCallum A, Schultz K, Singh S (2009) FACTORIE: Probabilistic programming via imperatively defined factor graphs. In: Proceedings of the 23rd Annual Conference on Neural Information Processing Systems (NIPS-09)

Milch B, Marthi B, Russell SJ, Sontag D, Ong DL, Kolobov A (2005) Blog: Probabilistic models with unknown objects. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)

Moldovan B, De Raedt L (2014) Occluded object search by relational affordances. In: IEEE International Conference on Robotics and Automation (ICRA-14)

Moldovan B, Moreno P, van Otterlo M, Santos-Victor J, De Raedt L (2012) Learning relational affordance models for robots in multi-object manipulation tasks. In: IEEE International Conference on Robotics and Automation (ICRA-12)

Muggleton S (1996) Stochastic logic programs. In: De Raedt L (ed) Advances in Inductive Logic Programming, IOS Press, pp 254–264

Nitti D, De Laet T, De Raedt L (2013) A particle filter for hybrid relational domains. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-13)

Nitti D, De Laet T, De Raedt L (2014) Relational object tracking and learning. In: IEEE International Conference on Robotics and Automation (ICRA), June 2014

Pfeffer A (2001) IBAL: A probabilistic rational programming language. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)

Pfeffer A (2009) Figaro: An object-oriented probabilistic programming language. Tech. rep., Charles River Analytics

Poole D (2003) First-order probabilistic inference. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)

Richardson M, Domingos P (2006) Markov logic networks. Machine Learning 62(1-2):107–136

Santos Costa V, Page D, Cussens J (2008) CLP(BN): Constraint logic programming for probabilistic knowledge. In: De Raedt et al (2008), pp 156–188

Sato T (1995) A statistical learning method for logic programs with distribution semantics. In: Proceedings of the 12th International Conference on Logic Programming (ICLP-95)

Sato T, Kameya Y (2001) Parameter learning of logic programs for symbolic-statistical modeling. J Artif Intell Res (JAIR) 15:391–454

Sato T, Kameya Y (2008) New advances in logic-based probabilistic modeling by prism. In: Probabilistic Inductive Logic Programming, pp 118–155

Skarlatidis A, Artikis A, Filiopou J, Paliouras G (2014) A probabilistic logic programming event calculus. Theory and Practice of Logic Programming (TPLP) FirstView

Suciu D, Olteanu D, Ré C, Koch C (2011) Probabilistic Databases. Synthesis Lectures on Data Management, Morgan & Claypool Publishers

Taskar B, Abbeel P, Koller D (2002) Discriminative probabilistic models for relational data. In: Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI-02)

Thon I, Landwehr N, De Raedt L (2008) A simple model for sequences of relational state descriptions. In: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD-08)

Thon I, Landwehr N, De Raedt L (2011) Stochastic relational processes: Efficient inference and applications. Machine Learning 82(2):239–272

Van den Broeck G, Thon I, van Otterlo M, De Raedt L (2010) DTProbLog: A decision-theoretic probabilistic Prolog. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)

Van den Broeck G, Taghipour N, Meert W, Davis J, De Raedt L (2011) Lifted probabilistic inference by first-order knowledge compilation. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)

Vennekens J, Verbaeten S, Bruynooghe M (2004) Logic programs with annotated disjunctions. In: Proceedings of the 20th International Conference on Logic Programming (ICLP-04)

Vennekens J, Denecker M, Bruynooghe M (2009) CP-logic: A language of causal probabilistic events and its relation to logic programming. Theory and Practice of Logic Programming (TPLP) 9(3):245–308

Wang WY, Mazaitis K, Cohen WW (2013) Programming with personalized pagerank: a locally groundable first-order probabilistic logic. In: Proceedings of the 22nd ACM International Conference on Information and Knowledge Management (CIKM-13)